République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique
Département d'Electronique

Mémoire de fin d'études
En vue de l'obtention du diplôme d'ingénieur d'état en Electronique

Thème

Implémentation sur DSP TMS320C5000 de filtres optimaux appliqués aux images et introduction de réseaux neuronaux

Proposé et Dirigé par :

Dr L. HAMAMI (MC)
Melle A. MOUSSAOUI (CC)

Réalisé par:

Melle BOULFANI Yasmine
Melle DOUMANDJI Samah

Promotion Juin 2004

REMERCIEMENTS

En premier, nous adressons nos plus vifs remerciements à notre promotrice Dr L. HAMAMI pour les conseils avisés qu'elle nous a toujours prodigués, pour les connaissances dont elle nous a fait bénéficier, son suivi attentif et sa confiance qui nous a été très précieuse. Mais aussi nous tenons à la remercier pour nous avoir fait l'honneur de nous encadrer.

Nous remercions également notre co-promotrice Dr A. MOUSSAOUI.

Nos remerciements les plus sincères vont au Pr D. BERKANI, pour nous avoir fait l'honneur de bien vouloir présider ce jury.

Il nous est agréable d'exprimer aussi nos remerciements à M R. BOUSSEKSOU, qui a bien voulu juger ce travail.

Nous tenons à remercier également l'ensemble des enseignants qui ont contribué à notre formation.

Nos sincères remerciements vont aussi à toutes les personnes qui, de prés ou de loin ont contribué à la réalisation de ce travail, en particulier D.DOUACHE, S. BENDI, et M.BRAHIM.

Nous voudrions enfin remercier en particulier toute la promotion d'électronique 2004.

Samah & Yasmine

Dédicace

A Dieu le tout puissant.

A mes très chers parents.

A mes chers frères et soeur : Amine, Hamza et Lamia.

A ma promotrice Mme L.Hamami.

A binomti Yasmine qui sans elle ce travail n'aurait pas été fait.

A tous mes amis de l'association TOUIZA,

A tous mes amis et amies de l'école polytechnique

Aux familles DOUMANDJI et DAHDAH.

Et à tous ceux qui me sont chers.

Je dédie ce modeste travail

Samah DOUMANDJI

Dédicace

A ma mère à ma mère à ma mère... 36 milliards de fois.

A mon très cher père.

A mes chères soeurs : Fadia, Nawel et en particulier Feriel.

A toute la famille BOULFANI

A ma promotrice Mme L.Hamami.

A mon binôme mamah qui sans elle ce travail n'aurait pas été fait.

A mes très chères et uniques amies Ikram et Nanou.

A tous mes amis de l'association de volontariat TOUIZA,

Et à tous ceux qui me sont chers.

Je dédie ce modeste travail

Yasmine BOULFANI

ملخص

يكمن هذا البحث في برمجة مرشحين محسنين-مستخلصين للمحيط-مطبقين على الصور, فوق الميكرو بروسسور المتخصصDSP , بدون و باستعمال شبكة عصبية إصطناعية .

المرشحان الإصطناعيان اللذان استعملناهما هما مرشحا شان و دريش,

و DSP المستعمل هو من جيل TMS320C5000.

فيما يخص الترشيح العصبي , توجه اختيارنا إلى الشبكة العصبية المتعددة الطبقات , بحيث تمت مرحلة التعلم بالبرنامج التطبيقيMATLAB.

الكلمات المفتاحية الصورة, الترشيح التحسيني, الشبكات العصبية ,DSP.

Résumé

Ce travail consiste en une implémentation de filtres optimaux (détecteurs de contours) appliqués aux images, sur un microprocesseur spécialisé « DSP », sans et avec utilisation des réseaux de neurones.

Les filtres optimaux choisis sont ceux de Shen-Castan et de Deriche, et le DSP utilisé est celui de la génération TMS320C5000.

Pour le filtrage neuronal, nous avons opté pour le réseau de neurones multicouche dont l'apprentissage a été fait sous MATLAB.

Mots clés : Image, filtre optimal, réseaux de neurones, DSP.

Abstract

This work consists to an implementation of optimal filters (Edges detectors) applied to images, on a specialized microprocessor "DSP", without and with using neuronal networks.

The chosen optimal filters are those of Shen-Castan and Deriche, and the DSP used is the one of the TMS320C5000 generation.

For the neuronal filtering, we opted for multilayer neuronal network whose training (learning) has been made with MATLAB.

Key words: Image, optimal filter, neuronal networks, DSP.

Table des matières

Chapitre III : Les DSPs TMS320C5000 58

Chapitre IV : Travail effectué 83

Chapitre I : Traitement d'images

Chapitre II : Réseaux de neurones

Chapitre III : Les DSPs TMS320C5000

Chapitre IV : Travail effectué

Chapitre V : Résultats et interprétations

Chapitre VI : Description de l'interface

Introduction générale

L'homme s'est toujours servi de représentations graphiques pour transmettre des connaissances ou pour noter des situations dont il voulait garder le souvenir. L'utilisation des images se justifie par le fait que le contenu représenté peut être facilement compréhensible par de larges classes de personnes possédant une culture analogue.

Le traitement d'images a pour vocation l'étude, la conception et la réalisation de systèmes d'exploitation d'images considérées comme véhicules de l'information. Son but ultime consiste à extraire le contenu informationnel (ou information pertinente) des images en vue d'une prise de décision ou d'une action. Cette information peut être à deux niveaux différents :

- Le traitement de « bas niveau » auquel est affectée la tâche d'extraction des primitives pertinentes de l'image dont le but est la réduction de la quantité d'information contenue dans l'image.
- Le traitement de « haut niveau » destiné à l'interprétation du contenu de l'image dans un but de reconnaissance et de compréhension.

Une nouvelle technique est apparue. Elle consiste à utiliser l'image comme un exemple ou un ensemble d'exemples qui servent d'entrées à un réseau de neurones artificiels. Ce dernier s'inspire essentiellement du processus de traitement de l'information effectué par le cerveau humain.

Ainsi, notre travail s'inscrit dans le cadre d'un projet dirigé par le Dr L.Hamami, dont l'objectif consiste en une implémentation de filtres optimaux (détecteurs de contours) appliqués aux images, et une amélioration par utilisation des réseaux de neurones. Ces filtres doivent être implémentables sur un matériel spécialisé : un processeur dédié signal « DSP ».
Pour cela nous devons prendre connaissance de la structure de l'image, des traitements qu'elle doit subir et plus précisément du filtrage optimal, des réseaux de neurones et du processeur le plus approprié.

Les filtres utilisés sont ceux de Shen-Castan et de Deriche. Après étude et analyse des moyens disponibles, nous avons opté pour le réseau multicouche et l'utilisation d'un DSP de la firme Texas Instruments (TMS320C5000).

Ce mémoire est organisé de la manière suivante :

- Le premier chapitre introduit des généralités et quelques définitions fondamentales relatives au domaine du traitement d'images. Aussi, nous aborderons les méthodes de détection de contours par les opérateurs optimaux.
- Le second chapitre sera consacré à l'étude des réseaux de neurones, au passage du modèle biologique au modèle artificiel, aussi nous donnerons le protocole d'apprentissage et les différentes classes et architectures de ces réseaux tout en détaillant le réseau multicouche.
- Dans le troisième, nous donnerons une présentation générale des DSPs, puis nous nous étalerons sur l'étude matérielle et logicielle du DSP TMS320C5000.
- Le quatrième chapitre sera consacré à la partie implémentation et mise en œuvre du filtrage optimal sans et avec l'utilisation des réseaux de neurones.
- Dans le cinquième chapitre nous présenterons les résultats obtenus et les interprétations correspondantes.
- Quant au dernier chapitre, il sera consacré à la description de l'interface utilisée.

Nous conclurons par une synthèse du travail effectué et quelques perspectives.

I.1. Introduction

Avec la parole, l'image a toujours constituée l'un des moyens le plus privilégié qu'utilise l'homme pour communiquer avec autrui. En effet c'est un moyen de communication universel dont la richesse du contenu permet aux êtres humains de tout âge et de toute culture de se comprendre. Chacun peut analyser l'image à sa manière, pour en dégager une impression et d'en extraire des informations précises.
De ce fait, le terme générique *traitement d'images* est l'ensemble des méthodes et techniques opérant sur celles-ci, dans le but de rendre cette opération possible, plus simple, plus efficace et plus agréable, d'améliorer l'aspect visuel de l'image et d'en extraire des informations jugées pertinentes en regard de l'application concernée. [6]
Avant de donner les notions fondamentales et les outils de traitement d'images, il serait utile de définir la notion d'image et les raisons de son traitement.

- **Qu'est ce qu'une image ?**

L'image est une représentation d'une personne ou d'un objet par la peinture, la sculpture, le dessin, la photographie, le film, etc.
C'est aussi un ensemble structuré d'informations qui, après affichage sur écran, ont une signification pour l'œil humain.
Afin que l'image puisse être traitée par la machine, sa numérisation est nécessaire. [6]
Nous distinguons trois groupes d'images : [1]

1. Les images *physiques visibles*, qui sont parfaitement matérielles et de nature volatile (soit des images optiques, constituées de photons dans le domaine visible, soit les images électro-optiques), ou de nature permanente (reproductions de toutes sortes : clichés photographique, peintures, sculptures, documents, imprimés, etc.
2. Les images *physiques non visibles*, les images optiques hors du domaine visible ou images de nature immatérielle (spectres de physique, cartes de population, de températures, etc.). Une vue infrarouge est une image non visible mais, après impression d'une pellicule sensible à ce rayonnement, elle devient une image physique visible.
3. Les images *mathématiques*, qui sont des concepts et sont donc invisibles par nature. Elles peuvent être continues (par exemple une fonction $f(x, y)$) ou discrètes : ce sont alors des tableaux ou matrices de nombres.

● Pourquoi et comment traiter les images ?

On peut dire que l'on traite des images dés lors que l'on extrait une information de cette image. Il existe de très nombreuses façons de traiter des images. Une première manière de tenter une classification des types de traitement est de se référer au but poursuivi.

Pourquoi cherche-t-on à traiter une image ? [1]

1. On peut chercher à améliorer sa qualité subjective (la rendre plus agréable à l'œil) ou objective, donc mesurable : on veut améliorer son contraste, accroître la perception de certains détails ou contours, faire ressortir plus nettement certaines zones ou certaines formes, diminuer le bruit, etc. Il s'agit donc de techniques d'*amélioration* qui font appel à des procédés variés créant des lissages, rehaussements, etc.
2. On peut chercher à trouver une image « idéale » de l'objet qui a été dégradé par divers processus : on veut diminuer les fluctuations dues à des phénomènes de turbulence. Il s'agit là de techniques de *restauration.*
3. On peut chercher à détecter la présence de certaines formes, certains contours ou certaines textures de modèle connu, sans vouloir préserver les autres informations.
4. On peut chercher à réduire l'énorme quantité d'informations contenues dans une image, à la « comprimer » afin de gagner en vitesse de transmission, en encombrement des organes utilisés, en capacité de stockage, etc. tout en ne dégradant que le moins possible les images considérées. Ce domaine est celui du *codage*, de la *compression* des données et de l'*approximation* des images.
5. On peut chercher à faire une analyse de l'image. Cette dernière consiste en l'extraction des informations contenues dans les divers objets de la scène, sans toutefois fournir une interprétation. Il s'agit des techniques d'*extraction d'attributs*, c'est-à-dire de « formes » (contours, textures) et de *segmentations.*

Durant ce chapitre nous verrons quelques notions fondamentales, ainsi que quelques outils de traitement d'images.

I.2. Notions fondamentales en traitement d'images

I.2.1. Pixel

Abréviation de « Picture Element », mot anglo-américain de **pix** pour pics, abréviation de « picture » (image) et de **el** pour « element ». Élément d'image, le pixel est le plus petit point de l'image, c'est une entité calculable qui peut recevoir une structure et une quantification. La lettre A, par exemple, peut être affichée comme un groupe de pixels dans la figure ci-dessous :

La quantité d'information que véhicule chaque pixel donne des nuances entre images monochromes et images couleurs. Dans le cas d'une image monochrome, chaque pixel est codé sur un octet, et la taille mémoire nécessaire pour afficher une telle image est directement liée à la taille de l'image.

Dans une image couleur (R.V.B.), un pixel peut être représenté sur trois octets : un octet pour chacune des couleurs : rouge (R), vert (V) et bleu (B). [6]

I.2.2. Voisinage d'un pixel

Le voisinage d'un pixel est composé de tous les pixels qui l'entourent immédiatement. Si p est un pixel d'une image D, alors le voisinage de p est le plus petit sous-ensemble de D qui contient p.

Dans une image numérique, on distingue deux types de connexités relatives au voisinage utilisé : la 4-connexité (Figure I.1.a) et la 8-connexité (Figure I.1.b).

Le voisinage d'un pixel (i,j) est dit 4-connexe s'il est formé des quatre pixels de coordonnées spatiales (i+1,j),(i,j+1),(i-1,j),(i,j-1), et il est dit 8-connexe s'il est formé des pixels de coordonnées spatiales (i+1,j),(i,j+1),(i-1,j),(i,j-1),(i+1,j-1),(i+1,j+1),(i-1,j+1),(i-1,j-1). [9]

a) Voisinage 4-connexes **b) Voisinage 8-connexes**

Figure I.1 : Voisinage d'un pixel.

I.2.3. Image numérique

Contrairement aux images obtenues à l'aide d'un appareil photo, ou dessinées sur du papier, les images manipulées par un ordinateur sont numériques (représentées par une série de bits).

L'image numérique est l'image dont la surface est divisée en éléments de tailles fixes appelés cellules ou pixels, ayant chacun comme caractéristique un niveau de gris ou de couleurs prélevé à l'emplacement correspondant dans l'image réelle, ou calculé à partir d'une description interne de la scène à représenter.

La numérisation d'une image est la conversion de celle-ci de son état analogique (distribution continue d'intensités lumineuses dans un plan xOy en une image numérique représentée par une matrice bidimensionnelle de valeurs numériques $f(x,y)$ où :

x, y : coordonnées cartésiennes d'un point de l'image.

$f(x, y)$: niveau de gris en ce point

Pour des raisons de commodité de représentation pour l'affichage et l'adressage, les données images sont généralement rangées sous formes de tableau I (matrice) de N lignes et P colonnes. Chaque élément $I(x ,y)$ représente un pixel de l'image et à sa valeur est associé à un niveau de gris codé sur m bits (2^m niveaux de gris ; 0 = noir ; 2^m-1 = blanc). La valeur en chaque point exprime la mesure d'intensité lumineuse perçue par le capteur. [6]

$$I=\begin{pmatrix} f(0,0) & f(0,1) & \cdots & \cdots & f(0,P-1) \\ f(1,0) & f(1,1) & & & f(1,P-1) \\ \cdots & \cdots & & & \cdots \\ \cdots & \cdots & & & \cdots \\ f(N-1,0) & \cdots & & & f(N-1,P-1) \end{pmatrix}$$

I.2.4. Résolution d'une image

La résolution d'une image est définie par un nombre de pixels par unité de longueur de la structure à numériser (classiquement en dpi (dots per inches) ou ppp (points par pouce)). Ce paramètre est défini lors de la numérisation et dépend principalement des caractéristiques du matériel utilisé lors de processus de numérisation. Plus le nombre de pixels est élevé par unité de longueur de la structure à numériser, plus la quantité

d'information qui décrit cette structure est importante et plus la résolution est élevée. La résolution d'une image numérique définit le degré de détail qui va être représenté sur cette image. [9]

I.2.5. Image en niveaux de gris

Dans une image en niveaux de gris, la couleur d'un pixel peut prendre des valeurs allant du noir au blanc, en passant par un nombre fini de niveaux intermédiaires. Dans une telle image les intensités du rouge, du vert, et du bleu de chaque pixel sont égales. En général, les images en niveaux de gris renferment 256 teintes de gris (image à 256 couleurs), simplement chacune de ces 256 couleurs est définie dans la gamme des gris. Par convention la valeur zéro représente le noir (intensité lumineuse nulle) et la valeur 255 le blanc (intensité lumineuse maximale). Chaque pixel n'est donc plus représenté par un bit, mais par un octet. Le nombre de niveaux de gris dépend du nombre de bits utilisés pour décrire la " couleur " de chaque pixel de l'image. Plus ce nombre est important, plus les niveaux possibles sont nombreux. [6] [9]

Valeurs des niveaux de gris et teintes de gris correspondantes
0....20....40....50....80....100....120....140....160....180....200....220....240....255

Figure I.2 : Valeurs des niveaux de gris et teintes de gris correspondantes.

Exemple :

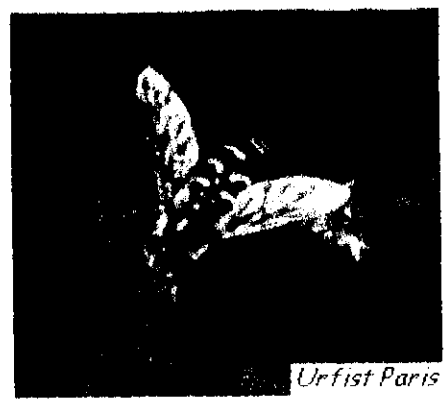


Figure I.3 : Exemple d'une image en niveaux de gris.

I.2.6. Image binaire

Ces images sont constituées de points (pixels) qui ne peuvent avoir que deux états : noir ou blanc. On les appelle des images « au trait ». On trouve parmi elles des pages de texte, des signatures, des plans, des dessins, etc. [2]

Exemple:

Figure I.4 : Image noir & blanc « au trait ».

I.2.7. Image en couleurs

Une image en couleurs est censée représenter le mieux possible la réalité. La représentation des couleurs s'effectue de la même manière que les images monochromes avec cependant quelques particularités. En effet, il faut tout d'abord choisir un modèle de représentation. On peut représenter les couleurs à l'aide de leurs composantes primaires. Les systèmes émettant de la lumière (écrans d'ordinateurs,...) sont basés sur le principe de la synthèse additive : les couleurs sont composées d'un mélange de rouge, vert et bleu (modèle R.V.B.). [6] [4]

- ***Représentation en couleurs réelles (Images en "vraies couleurs" (ou 24 bits))***

Elle consiste à utiliser 24 bits pour chaque point de l'image. Huit bits sont employés pour décrire la composante rouge (R), huit pour le vert (V) et huit pour le bleu (B). Il est ainsi possible de représenter environ 16,7 millions de couleurs différentes simultanément. Cela est cependant théorique, car aucun écran n'est capable d'afficher 16 millions de points. Dans la plus haute résolution (1600 x 1200), l'écran n'affiche que 1 920 000 points. Par ailleurs, l'œil humain n'est pas capable de distinguer autant de couleurs.

Valeur R	Valeur V	Valeur B	Couleur correspondante	Commentaires
0	0	0		noir
0	0	1		un peu moins noir (nuance impossible à

				détecter à l'oeil par rapport au noir)
...	...	...	...	...
0	0	255		bleu
...	...	...	...	...
0	255	0		vert
...	...	...	...	...
255	0	0		rouge
...	...	...	...	...
128	128	128		couleur intermédiaire correspondant à un gris
255	255	255		blanc

Tableau I.1 : Représentation en vraie couleurs (24 bits).

L'information couleur de chaque pixel est donc codée par 3 octets, ce qui fait que les images en vraies couleurs sont des images très "lourdes".

Exemple :

Figure I.5 : Image numérisée avec 16 millions de couleurs.

➢ Représentation en couleurs indexées

Afin de diminuer la charge de travail nécessaire pour manipuler des images en 24 bits, on peut utiliser le mode de représentation en couleurs indexées. Le principe consiste à déterminer le nombre de couleurs différentes utilisées dans l'image, puis à

créer une table de ces couleurs en attribuant à chacune une valeur numérique correspondant à sa position dans la table.

La table, appelée palette, comporte également la description de chacune des couleurs, sur 24 bits.

Exemple :

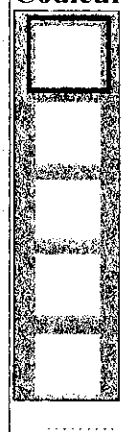
Palette avec les codes RVB	**Couleurs correspondantes:**
Couleur 0: 255 255 255	
Couleur 1: 255 255 204	
Couleur 2: 255 255 153	
Couleur 3: 255 255 102	
Couleur 4: 255 255 51	
Couleur 5:255 255 0	
.....	

Tableau I.2 : Exemple d'une palette de couleur.

Ainsi, un pixel à qui sera affectée la couleur numéro 0 sera visualisé en blanc, un pixel à qui sera affectée la couleur numéro 4 sera visualisé en jaune...

➢ Autres modèles de représentation

Le modèle R.V.B. représentant toutes les couleurs par l'addition de trois composantes fondamentales, n'est pas le seul possible. Il en existe de nombreux autres. L'un d'eux est particulièrement important. Il consiste à séparer les informations de couleurs (chrominance) et les informations d'intensité lumineuse (luminance). Il s'agit du principe employé pour les enregistrements vidéo. La chrominance est représentée par deux valeurs (selon des modèles divers) et la luminance par une valeur.

I.2.8. Bruit dans l'image

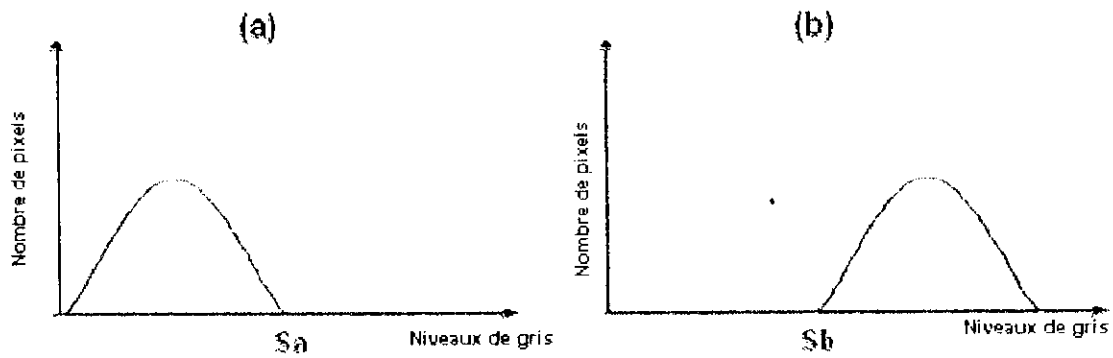
Le bruit dans une image est défini comme étant un phénomène de brusque variation d'un pixel isolé par rapport à ses voisins, il affecte la qualité de l'image et il provient soit du dispositif d'acquisition (scanner, caméra, amplification, quantification...) soit de la scène elle-même (poussières, rayures...). [9]

I.2.9. Histogramme

L'histogramme des niveaux de gris ou des couleurs d'une image est une fonction qui donne la fréquence d'apparition de chaque niveau de gris (couleur) dans l'image. Pour diminuer l'erreur de quantification, pour comparer deux images obtenues sous des éclairages différents, ou encore pour mesurer certaines propriétés sur une image, on modifie souvent l'histogramme correspondant. [6]

Il permet de donner un grand nombre d'informations sur la distribution des niveaux de gris (couleur) et de voir entre quelles bornes est repartie la majorité des niveaux de gris (couleur) dans les cas d'une image trop claire ou d'une image trop foncée. (Figure I.6) [9]

Il peut être utilisé pour améliorer la qualité d'une image (Rehaussement d'image) en introduisant quelques modifications, pour pouvoir extraire les informations utiles de celle-ci.



a) trop foncé **b) trop claire**

Figure I.6 : Histogramme d'image.

I.2.10. Contraste

C'est l'opposition marquée entre deux régions d'une image, plus précisément entre les régions sombres et les régions claires de cette image. Le contraste est défini en fonction des luminances de deux zones d'images. [6]

Si I_1 et I_2 sont les degrés de luminosité respectivement de deux zones voisines A_1 et A_2 d'une image, le contraste C est défini par le rapport :

$$C = \frac{I_1 - I_2}{I_1 + I_2} \quad \ldots\ldots(1)$$

I.3. Formats des images

A la différence d'une photographie ou d'un tableau, qui sont à la fois objet de la perception et mémoire de l'information. L'image numérique en tant qu'objet de perception n'existe pas, seule sa représentation est pertinente.

L'image numérique fixe (par opposition à la vidéo numérique ou l'animation en images de synthèse) se répartissent en deux catégories : image bitmap et image vectorielle. [2] [5]

I.3.1. L'image bitmap

L'image bitmap, dite aussi image matricielle, est en effet composée d'une matrice de points, les pixels. Ces points peuvent être en noir & blanc ou en couleur.

La caractéristique fondamentale d'une image bitmap est de représenter le plus exactement possible la réalité : elle vise au photoréalisme, elle est souvent produite par un périphérique de saisie (scanner, appareil photo numérique ou caméra). Elle sera le plus souvent (plus ou moins) retouchée.

● Organisation d'un format graphique bitmap

Une image bitmap est donc un ensemble de points (les pixels) assemblés sous forme de rectangle. Ces points sont « ordonnés » sur le plan informatique pour former un fichier bitmap.

Pour une représentation convenable, on peut stocker l'image dans un fichier qui se compose de deux parties : l'entête et les données de l'image.

Dans l'entête, on retrouve les informations indispensables (largeur de l'image en pixels, hauteur en nombre de lignes, nombre de bits pour coder un pixel), et les informations supplémentaires éventuelles (type de codage, numéro de version et signature du logiciel, abscisse et ordonnée d'origine à l'écran, palette de couleurs, etc.).

● Formats de fichiers bitmap

Les formats de fichiers bitmap sont très nombreux, seuls quelques-uns sont utilisés couramment, et quelques autres par les spécialistes de la PAO ou pour des domaines d'activité très spécifiques. (Voir annexe 1).

I.3.2. L'image vectorielle

A l'inverse de l'image bitmap, une image vectorielle est le plus souvent créée sans rapport avec la réalité. Elle peut avoir pour objectif de représenter la réalité, mais elle ne vise pas au photoréalisme.

Une image vectorielle n'est pas composée de points. Elle est formée de courbes, d'objets..., dotés ou non de contour et de surface.

Si l'image bitmap est une photo, l'image vectorielle est un dessin.

Contrairement à l'image bitmap, l'image vectorielle n'a pas obligatoirement la forme d'un rectangle. Si on représente un cercle en mode vectoriel, le fichier aura la forme d'un cercle.

● Formats de fichiers vectoriels

A l'inverse des images bitmap qui sont photoréalistes et encombrantes, les images ou illustrations vectorielles sont beaucoup moins encombrantes en ce qui concerne la taille des fichiers générés, mais elles ne peuvent en aucun cas se prévaloir d'un quelconque aspect photoréaliste.

Les formats de fichiers vectoriels sont moins nombreux que les formats de fichiers bitmap. (Voir annexe 1).

I.4. Étapes de l'analyse d'images

La plupart des applications de traitement d'image passent par les étapes suivantes : [3]

- ***Acquisition*** : Echantillonnage, Quantification.
- ***Analyse globale de l'image et transformations ponctuelles*** : Histogramme ; statistiques (moyenne, écart-type, etc.) ; transcodage (palette de couleur [*LUT*]) et classification.
- ***Opérations entre images*** : Indices, ratio, différence, opérations logiques, masques et seuillage.
- ***Amélioration, filtrage et segmentation*** : Opérations de convolution (lissage, rehaussement, détection de contours) ; squelettisation ; vectorisation.
- ***Interprétation et sémantique*** : Cartographie thématique (classification automatique et supervisée) ; cartographie vecteur ; représentation des graphes et de la topologie.

I.5. Domaines d'applications du traitement d'images

Le traitement d'images possède l'aspect multidisciplinaire. On trouve ses applications dans des domaines très variés tels que les télécommunications (T.V., vidéo, publicité,...), la médecine (radiographie, ultrasons,...), la biologie, l'astronomie, la géologie, l'industrie (robotique, sécurité), la météorologie, l'architecture, l'imprimerie, l'armement (application militaire).

De nouvelles applications pratiques sont possibles aujourd'hui et touchent tous les domaines d'activités, tels que : métiers du spectacle, de la radio, créations artistiques, etc. [6]

I.6. Outils de traitement d'images

La qualité d'une image peut être dégradée pour des raisons diverses. Les images brutes permettent rarement de parvenir à une extraction directe des objets à analyser :

- soit parce que l'éclairement de l'objet n'est pas uniforme,
- soit parce que le contraste n'est pas suffisant,
- soit encore parce que l'objet est perçu à travers un bruit assez important : les images contiennent donc un signal et du bruit (dont on veut éliminer la plus grande partie possible).

Avant d'extraire les objets et d'analyser une image, il est donc souvent nécessaire d'améliorer sa qualité en lui appliquant quelques traitements (par exemple : modification d'histogramme, binarisation, amélioration du contraste, élimination de bruits, etc.).

Les traitements s'appliquent toujours aux images en niveau gris et parfois aussi sur des images couleurs. [12]

Nous allons présenter dans ce qui suit quelques types de ces traitements.

I.6.1. Modification d'histogramme

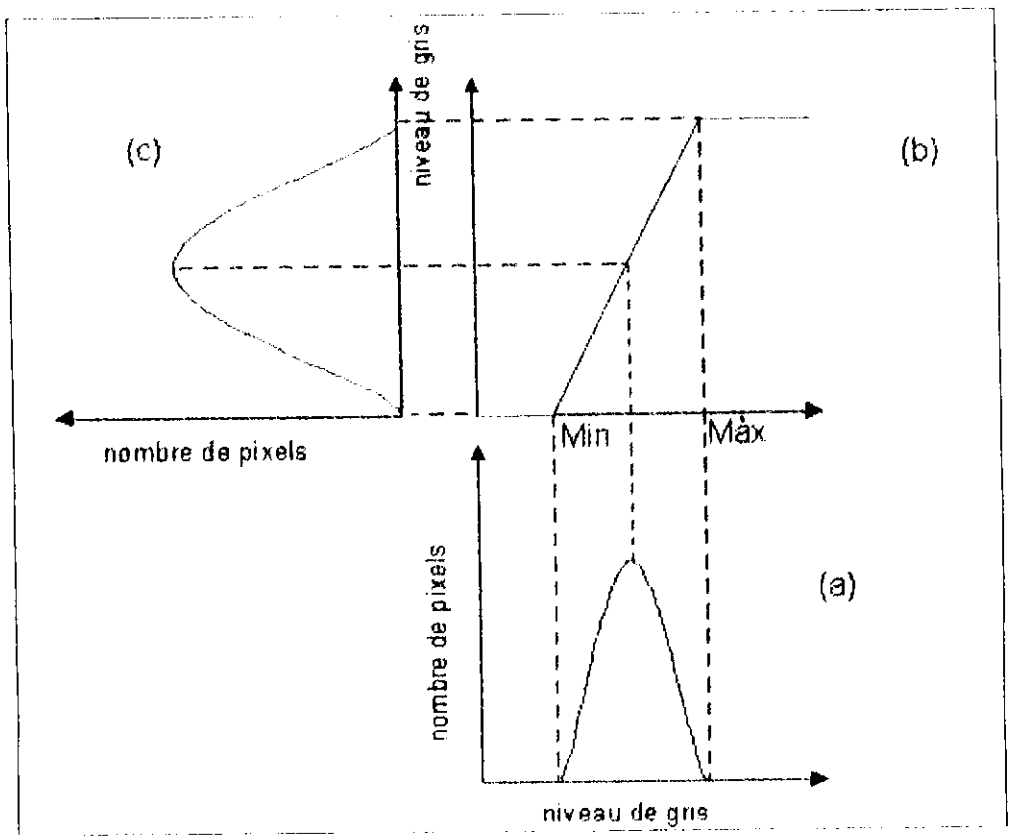
L'histogramme d'une image est une fonction qui donne la fréquence d'apparition de chaque niveau de gris (ou couleur) dans l'image, (voir I.2.9). Or parfois certaines images apparaissent trop foncées (les niveaux de gris de l'image sont tassés vers le haut de l'échelle), ou trop claires (les niveaux de gris de l'image sont tassés vers le bas de

l'échelle), ou bien peu contrastées (les niveaux de gris de l'image sont regroupés dans un intervalle étroit).

De ce fait, afin d'améliorer l'image on est ramené à modifier son histogramme, qui a pour but de redistribuer les niveaux de gris de l'image afin de leur faire occuper toute la bande de nuance possible. (Figure I.7).

Cette méthode est basée sur les *transformations ponctuelles* d'intensité. C'est-à-dire, qu'à tout pixel d'intensité a_s on associe une intensité $a_s' = T(a_s)$.

Du fait de leur caractère ponctuel, les méthodes de transformation d'histogramme n'affectent pas la forme des régions. Elles en modifient uniquement l'apparence visuelle. [9] [7]



a)- Histogramme de l'image originale.
b)- Fonction de transformation.
c)- Histogramme recadré.

Figure I.7 : Principe de la modification d'histogramme.

I.6.2. Binarisation (par seuillage)

Binariser une image en niveaux de gris consiste à la transformer en une image biniveaux, c'est-à-dire attribuer à chaque pixel de l'image la valeur 0 « noir » ou 255 « blanc ». La dynamique de l'image est alors réduite à deux luminosités.

La mise en oeuvre la plus simple est de fixer un seuil ajustable, identique pour toute l'image. Pour une échelle à 256 niveaux de gris, cette valeur est comprise entre 0 et 255, généralement dans la plage médiane, chaque pixel est comparé à ce seuil en polarité directe, ceux de niveaux de gris inférieurs sont mis à « blanc » (0), ceux supérieurs mis à « noir » (1).

La difficulté réside dans le choix du seuil. Si le seuil est fait par le scanner un opérateur peut effectuer un réglage en fonction de l'aspect visuel du document et du résultat d'une première acquisition. Pour le seuillage automatique d'une image à niveaux de gris, une méthode consiste à tracer un histogramme de ces niveaux de gris, et choisir le seuil au fond de la vallée qui sépare le pic correspondant au niveau de gris du fond et le premier pic suivant (Figure I.8). [8] [13]

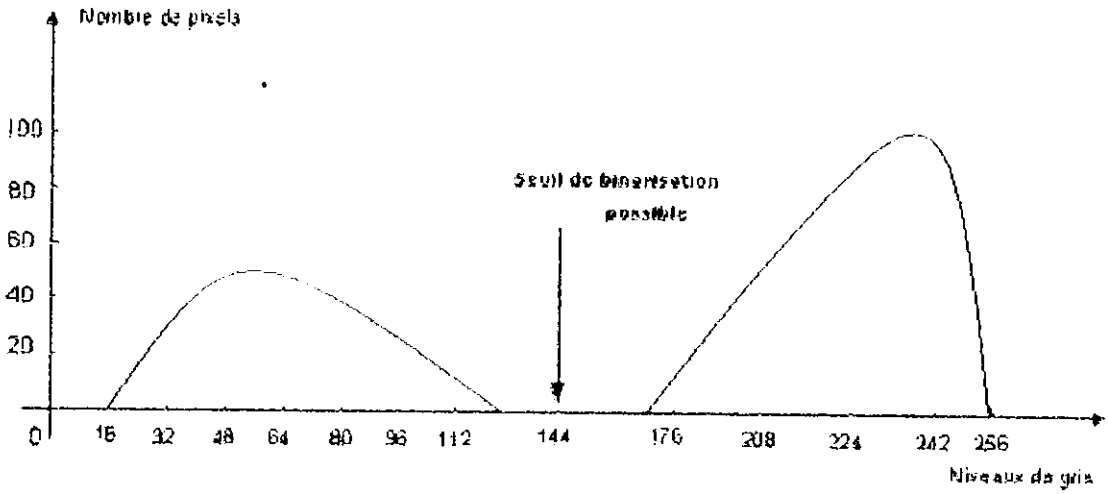


Figure I.8 : Choix du seuil sur l'histogramme de l'image en niveaux de gris.

I.6.3. Amélioration du contraste

Ce traitement ne s'applique qu'aux images en niveaux de gris.

On considère qu'une image est bien contrastée si les luminosités des pixels de l'image sont au mieux réparties sur l'axe [0,255].

La façon la plus directe consiste à normaliser les luminosités des pixels à partir des maximum et minimum des luminosités dans l'image originale. [13] [42]

Soit m la luminosité minimale et M la luminosité maximale dans l'image originale. Il

s'agit d'effectuer une transformation affine de chaque luminosité pour ramener la dynamique de [m;M] à [0;255].

Nous pouvons citer le cas du rehaussement d'images :

Rehaussement d'images

Le but des procédures de rehaussement d'image est d'améliorer leur qualité visuelle, ou de les convertir à une forme qui facilite leur interprétation soit par l'homme ou par la machine. Puisque la qualité visuelle d'une image est une notion subjective, toutes ces méthodes et techniques sont interactives et permettent ainsi à l'analyste, par essai et erreur, de transformer les images et d'adapter leur contenu en fonction de ses propres critères de qualité visuelle.

Les opérations de rehaussement de contraste peuvent être décrites de la façon suivante:

$$f_s(i,j) \quad T_g f_e(i,j)$$

Avec :

$f_s(i,j)$: Le niveau de gris d'un pixel de l'image de sortie.

$f_e(i,j)$: Le niveau de gris du même pixel de l'image d'entrée.

T_g : Fonction mathématique de transformation

Si la fonction mathématique agit seulement par modification de l'intervalle des niveaux de gris de l'image d'entrée, on parle d'un rehaussement par **étirement du contraste**. Si la fonction de transformation est calculée de sorte que non seulement l'intervalle soit modifié mais aussi que l'histogramme des niveaux de gris de l'image de sortie prenne une forme particulière, on parle de rehaussement par **modification de l'histogramme**.

I.6.4. Élimination de bruits

Le transfert de l'image d'un objet jusqu'à l'ordinateur lors de l'acquisition des images se produit avec un certain bruit. Le bruit est dû en particulier aux imperfections de la source qui génère l'image, au capteur proprement dit (caméra, détecteur électronique, etc.), et enfin à toute l'électronique de transfert. [12]

L'amélioration de l'image est essentiellement obtenue par ce que l'on appelle une opération de *filtrage*. Il existe un grand nombre de filtres, que l'on peut classer en 4 grandes catégories :

- Les filtres linéaires.
- Les filtres non linéaires.
- Les filtres morphologiques.
- Les filtres optimaux.

● Filtres linéaires

La transformation de chaque pixel est le fruit d'une combinaison linéaire des pixels voisins.

Ces opérateurs sont caractérisés par leur réponse impulsionnelle $h(x,y)$. [8]

La relation entrée-sortie est décrite par l'équation de convolution :

$$C(x,y) = A * h(x,y) = \iint A(\alpha,\beta)h(x-\alpha, y-\beta)d\alpha d\beta \quad \ldots\ldots\ldots\ldots (2)$$

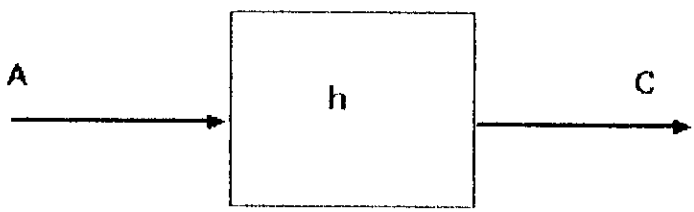


Figure I.9 : Filtrage linéaire.

Dans le cas d'une image numérique, l'équation (2) est remplacée par la relation de convolution discrète :

$$D[i,j] = \sum_m \sum_n h[m,n]A[i-m, j-n] = \sum_m \sum_n A[m,n]h[i-m, j-n] \quad \ldots\ldots\ldots\ldots (3)$$

Il existe une grande variété de filtres plus ou moins utilisés et il serait fastidieux de vouloir tous les décrire, notre choix sera porté sur les filtres linéaires les plus courants, parmi eux on définit :

- Le filtre moyen.
- Le filtre gaussien.

- **Filtre moyen**

Il consiste à considérer chaque point de l'image et d'en faire la moyenne avec les huit pixels qui lui sont voisins. Ceci va avoir pour effet d'adoucir l'image en réduisant les fluctuations des niveaux de gris.

Ce type de filtre utilisant la moyenne non pondérée des voisins peut être mis sous la forme d'un masque tel que celui-ci :

$$H_1 = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

On va alors déplacer ce masque sur toute l'image, le pixel affecté par la transformation étant le pixel central du masque. Le facteur 1/9 est égal à la somme des coefficients du masque et sert à normaliser le filtre de manière à ce que celui-ci n'influe pas sur l'intensité globale de l'image.

Si nous posons I_B comme étant l'image de départ (image bruitée), I_f l'image résultat (image filtrée), toutes deux de taille p*p, et H_1 le masque, nous avons alors pour chaque pixel de coordonnées x, y :

$$I_f(i,j) = \frac{1}{9}\sum_{i=-1}^{1}\sum_{j=-1}^{1} H_1(i+1, j+1)\, I_B(x+i, y+j) \quad \ldots\ldots\ldots\ldots (4)$$

D'autres filtres ont été réalisés en utilisant des coefficients de pondérations différents. Par exemple :

$$H_2 = \frac{1}{10}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad H_3 = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

le filtre H_2 donne plus de poids au pixel central, ce qui le rend moins actif que le filtre H_1, le filtre H_3 privilégie les directions x et y.

- **Filtre gaussien**

Il représente un filtre linéaire parmi les plus courants, tant par sa facilité de mise en œuvre que par la bonne qualité de ses résultats. Ce filtre tire son nom de la valeur de ses coefficients qui sont ceux d'une courbe de Gauss à deux dimensions.

Un filtre gaussien consiste en la convolution d'une image I_B avec une gaussienne $G(x,y,\sigma)$:

$I_F = G \otimes I_B$

Avec :

$$G(x,y,\sigma) = \frac{1}{\sqrt{2\prod\sigma}}\exp(-\frac{x^2+y^2}{2\sigma^2}) \quad \ldots\ldots\ldots\ldots\ldots\ldots(5)$$

Où G est un masque carré dont les coefficients sont les éléments discrétisés de la gaussienne.

Le principal avantage du filtre gaussien est qu'il présente le meilleur compromis entre les localisations spatiales et fréquentielles.

● Filtres non linéaires

Ce type de filtre s'oppose au précédent dans sa dénomination car il n'est pas le résultat d'une combinaison linéaire de pixels, les pixels voisins interviennent suivant une loi non linéaire.

Le domaine du filtrage non linéaire est extrêmement vaste, l'exemple le plus classique de filtre non linéaire est le filtre médian de Tuckey.

• Filtre médian

Son principe est celui de la figure I.10 et il peut être décomposé selon les étapes suivantes :

Pour chaque pixel de l'image

- On classe les pixels voisins du pixel courant (compris dans la fenêtre) par valeurs croissantes.
- On prend la valeur médiane des pixels classés et on l'affecte au pixel courant.

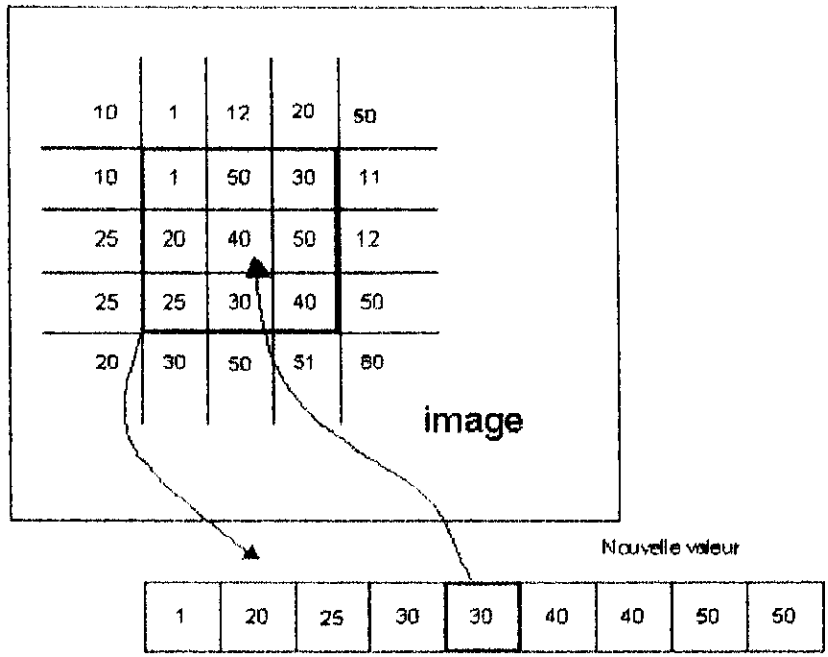


Figure I.10 : Principe de fonctionnement du filtre médian.

● Filtres morphologiques

Ce type de filtrage est réalisé par une combinaison de traitement linéaire et non linéaire et il est appliqué à des images binaires, chaque pixel de l'image peut être codé comme un élément logique donc à valeur *vraie* ou *fausse*, ainsi il est l'objet d'une transformation par une fonction logique plus ou moins complexe.

Parmi les opérateurs morphologiques (méthodes visant à améliorer les images binaires) nous citons : la dilatation, l'érosion, l'ouverture et la fermeture. [10]

• Dilatation

Elle permet d'éliminer les pixels blancs isolés. On effectue le **ET** logique des huit voisins du pixel considéré.

La dilatation élimine les tâches (les trous) blanches dans les zones noires mais ajoute des pixels noirs au contour des objets présents dans l'image. Donc la taille des objets dans l'image augmente.

• Erosion

Elle permet d'éliminer les pixels noirs isolés au milieu des parties blanches de l'image, on effectue le **OU** logique des huit voisins du pixel considéré.

En appliquant une érosion, ces tâches noires peuvent être éliminées mais la taille des objets présents dans l'image diminue car l'érosion enlève des pixels du contour, entraînant une déformation de certains objets.

- **Ouverture**

L'ouverture est constituée par une opération d'érosion suivie d'une dilatation. Elle permet de retrouver la taille normale des objets de l'image.

- **Fermeture**

La fermeture est l'opération inverse de l'ouverture, qui consiste à faire subir à l'image une dilatation suivie d'une érosion. Elle permet aussi de retrouver la taille normale des objets de l'image.

● Filtres optimaux

La segmentation des images est l'outil de base du traiteur d'images. Quelle que soit son origine, une image constitue une représentation d'un univers composé d'entités : objets dans une scène, cellules, organes du corps humain, routes... Le but de toute méthode de segmentation est l'extraction d'attributs caractérisant ces entités. Les attributs étudiés correspondent à des points d'intérêt ou à des zones caractéristiques de l'image : *contours* et *régions*, qui conduisent à deux approches très différentes.
L'approche étudiée dans notre travail sera une approche contour, utilisant les filtres de lissage et de dérivation de *Shen-Castan* d'une part et de *Deriche* d'autre part, en se basant sur l'étude de *Canny*. La détection de contours impliquera dans ce cas la recherche des discontinuités locales de la fonction des niveaux de gris de l'image en utilisant un filtrage récursif. [14]
Nous consacrerons donc à ce type de filtre une étude plus détaillée.
Tout d'abord nous allons présenter les différentes approches de détection de contours.

I.7. Filtrage optimal par approche contours

Détecter les contours dans une image, revient à localiser ses discontinuités. Pour cela, deux approches sont utilisées : *l'approche Gradient* et *l'approche Laplacien*. Ces approches se basent sur la différentiation de l'image en la dérivant une fois, on obtient alors le Gradient ou bien deux fois en obtenant ainsi le Laplacien. [14] [15]
La Figure I.11 montre les effets de la dérivation en présence d'un contour :

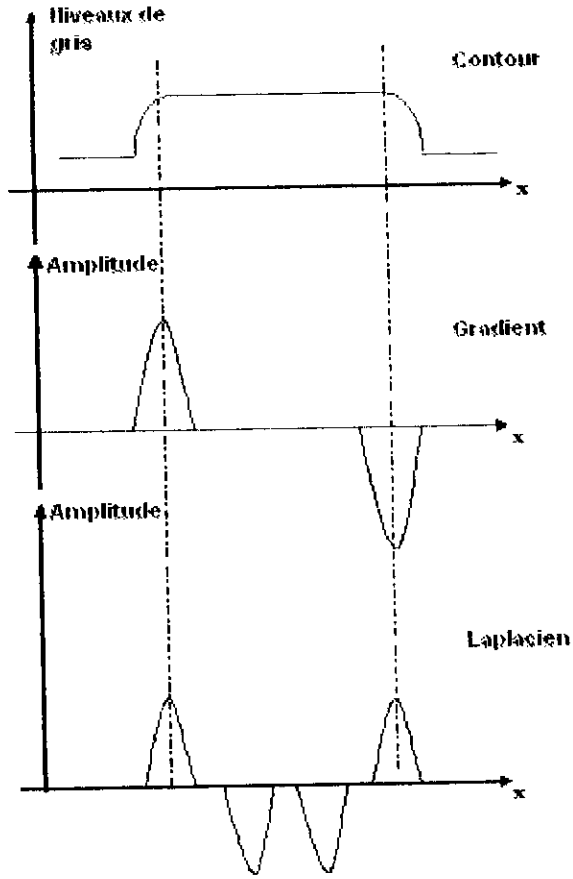


Figure I.11 : Détection de contours par dérivation.

I.7.1. Approche gradient

On peut ramener la détection de contours, sous réserve d'hypothèses raisonnables, au lissage et à la dérivation par filtrage linéaire et monodimensionnel. Soit *I(x,y)* une image de dimension 2. On considèrera que les contours correspondent aux discontinuités d'ordre 0 de *I*. La méthode consiste à :

- Calculer le gradient en chaque point de l'image,
- Créer l'image de la norme du gradient,
- Extraire les maxima locaux dans la direction du gradient,
- Effectuer un seuillage à effet hystérésis de l'image des maxima locaux.

Le vecteur gradient est défini au point *M(x, y)* par :

$$G(x,y) = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})^t = (I_x(x,y), I_y(x,y))^t \quad \ldots\ldots\ldots\ldots (6)$$

Avec t : la transposé du vecteur.

Le but du calcul du gradient est de déterminer la direction selon laquelle la variation locale des niveaux de gris est la plus forte (direction du gradient), ainsi que l'intensité de cette variation (norme du gradient).

La norme du gradient est donnée par : $$N(x,y)=\sqrt{I_x(x,y)^2+I_y(x,y)^2} \quad \text{.....(7)}$$

On détermine ensuite les points tels que $N(x,y)>s$, où s est un seuil fixé à priori.
Dans le cas d'images où la norme du gradient aux points de contour varie fortement selon les parties de l'image, cette méthode se révèle inefficace. En effet, on ne peut pas trouver un seuil unique permettant de traiter efficacement toutes les parties de l'image. Un moyen de contourner cette difficulté est d'extraire non pas les points de norme de gradient élevée, mais les maxima locaux de la norme du gradient (par exemple dans la direction de celui-ci). On élimine ensuite les points de norme de gradient faible avec un seuillage par hystérésis. Ce type de seuillage consiste à ne conserver que :

- Les points dont la norme du gradient est supérieure à un seuil haut (S_h).
- Les points dont la norme du gradient est supérieure à un seuil bas (S_b avec $S_b<S_h$) et appartenant à un bout de contour dont au moins un point possède une norme du gradient supérieure à S_h.

Ce type de seuillage permet de diminuer le nombre de bouts de contour non fermés, c'est-à-dire à obtenir des points de contour bien connectés entre eux.

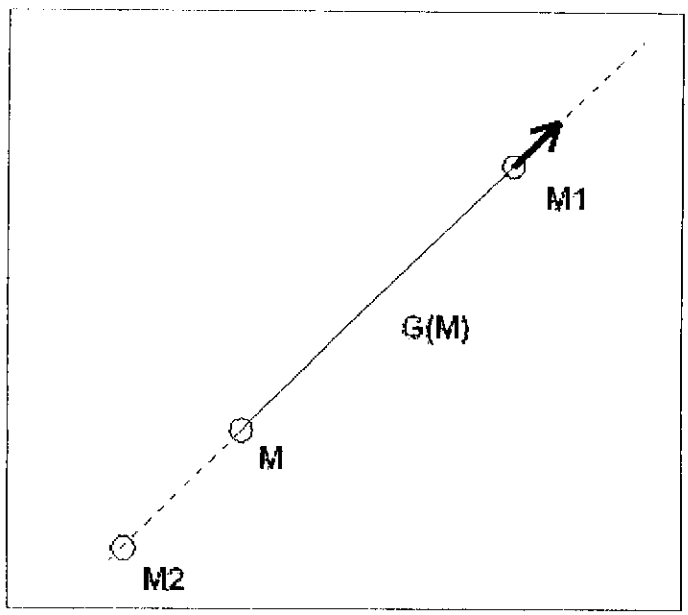


Figure I.12 : Extraction des maxima locaux. (Le point M est sélectionné si $N(M) > N(M_2)$ et $N(M) \geq N(M_1)$).

I.7.2. Approche Laplacien

Consiste à calculer la dérivée seconde de l'image . Les points de contour correspondent aux valeurs nulles du laplacien de l'image.

Le laplacien $L(x,y)$ d'un signal continu f est donné par :

$$L(x,y) = \frac{\partial^2 f(x,y)}{\partial^2 x} + \frac{\partial^2 f(x,y)}{\partial^2 y} \quad \ldots\ldots\ldots\ldots (8)$$

Donc détecter le contour d'une image revient à :

- Calculer son laplacien qui se ramène à effectuer le produit de convolution bidimentionnel de l'image avec l'un des masques suivants : H_1, H_2 et H_3.

$$H_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad , H_2 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad , \text{ et } H_3 = \begin{pmatrix} -1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}.$$

- Le contour est ensuite déterminé par la détection des passages par zéro du laplacien de l'image.
- Créer l'image des passages par zéros et de la norme du gradient.
- Effectuer un seuillage à effet hystérésis de l'image des maxima locaux.

I.7.3. Comparaisons entre les deux approches

L'utilisation du Gradient demande des calculs complexes mais donne de bons résultats, et pour ce qui concerne le Laplacien, il possède un inconvénient majeur qui est sa grande sensibilité au bruit. En effet, cet opérateur réalise une dérivée seconde de l'image et est donc très instable.

Des procédures de seuillage sont alors appliquées, de plus d'autres procédures sont utilisées après la détection de contour qui sont :

- La binarisation pour mettre en évidence les points de contour acceptés ou rejetés.
- L'amincissement du contour pour que celui- ci ait une épaisseur d'un seul pixel.
- La fermeture de contour qui consiste à prolonger les extrémités des contours qui ne sont pas fermés.

En conclusion, les contours des objets correspondent le plus souvent aux extrema locaux du Gradient ou aux zéros du Laplacien de la fonction des niveaux de gris. La

différenciation étant sensible au bruit, nous nous limiterons à la dérivation du premier ordre et donc à une approche gradient. Ainsi, les filtres de Deriche et Shen-Castan étudiés sont séparables (et peuvent donc se généraliser facilement à une dimension quelconque) et permettent une implémentation récursive.

I.7.4. Critères de performances et filtrage optimal

Le résultat d'une détection de contours peut bien sûr être évalué à l'oeil nu, mais l'appréciation est subjective. Les performances d'un détecteur se caractérisent essentiellement par :

- ***détection :*** l'opérateur doit donner une réponse au voisinage d'un contour,
- ***localisation :*** le contour doit être localisé avec précision,
- ***réponse unique :*** un contour doit provoquer une seule réponse de l'opérateur d'extraction.

a- Critères de Canny

Canny a introduit une modélisation définissant des critères évaluant :

1. ***La détection des points de contours*** : faible probabilité de ne pas détecter un vrai point de contour, et faible probabilité de marquer de faux points de contours. Ce critère correspond à maximiser le rapport signal sur bruit RSB :

$$\sigma = RSB = \frac{A}{\eta_0}\Sigma = \frac{A}{\eta_0}\frac{\int_{-\infty}^{0} f(x).dx}{\left(\int_{-\infty}^{+\infty} f^2(x).dx\right)^{1/2}} \quad \ldots\ldots\ldots\ldots(9)$$

2. ***La localisation des points de contours*** : les points marqués comme contours par le détecteur doivent être aussi près que possible du centre du contour véritable. Ce critère correspond à maximiser l'écart type de la position des passages par zéro, c'est à dire à l'inverse de l'espérance de la distance entre le vrai point de contour et le point de contour détecté :

$$\lambda = \frac{A}{\eta_0}\Lambda = \frac{A}{\eta_0}\frac{|f'(0)|}{\left(\int_{-\infty}^{+\infty} f'^2(x).dx\right)^{1/2}} \quad \ldots\ldots\ldots\ldots(10)$$

3. ***Réponse unique à un contour*** : le détecteur ne doit pas fournir de multiples réponses à un seul contour. On peut montrer que cela revient à minimiser l'expression :

$$x_{max} = \left| \frac{\int_{-w}^{w} f'^{2}(x).dx}{\int_{-w}^{w} f''^{2}(x).dx} \right|^{1/2} \quad \text{.....(11)}$$

Les deux premiers critères de détection et de localisation étant antinomiques, on les combine en maximisant le produit $\Sigma.\Lambda$ sous la contrainte du troisième critère. On obtient ainsi une équation différentielle dont la solution est :

$$\boxed{f(x) = a_1 e^{\alpha.x} \sin\omega.x + a_2 e^{\alpha.x} \cos\omega.x + a_3 e^{\alpha.x} \sin\omega.x + a_4 e^{-\alpha.x} \cos\omega.x + c} \quad \text{.....(12)}$$

b- Filtre de Deriche

Deriche, utilisant la même démarche que Canny, a cherché une réalisation de l'opérateur sous la forme d'un filtre RII (réponse impulsionnelle infinie). Il a abouti à la même équation différentielle (12). Seules les conditions aux limites sont différentes. Pour le filtre de Deriche, les conditions aux limites sont données par :

$$f(0) = 0, \quad f(+\infty) = 0, f'(0) = S, \text{ et } \quad f'(+\infty) = 0$$

On obtient alors : $a_1 = a_2 = a_4 = c = 0$. On les remplace dans l'équation (12) on obtient le filtre optimal :

$$\boxed{f(x) = \alpha e^{-\alpha x} \sin \omega.x} \quad \text{.....(13)}$$

En évaluant pour cet opérateur les différentes intégrales intervenant dans le calcul des critères de performance, on obtient les résultats suivants :

$$\Lambda = \sqrt{2\alpha} \qquad \Sigma = \sqrt{\frac{2\alpha}{\alpha^2 + \omega^2}}$$

En posant $\alpha = m\omega$, on obtient :

m	Λ	Σ	$\Sigma\Lambda$	x_{max}
$m>>1$	$\sqrt{2\alpha}$	$\sqrt{2\alpha}$	2	0.44
$m<<1$	$\sqrt{2\alpha}$	Λ / m	$2m$	1
$m=1$	$\sqrt{2\alpha}$	$\sqrt{1/\alpha}$	$\sqrt{2}$	0.58
$m=\sqrt{3}$	$\sqrt{2\alpha}$	$\sqrt{3/(2\alpha)}$	$\sqrt{3}$	0.5

On voit que pour des valeurs de x_{max} identiques, l'opérateur de Deriche présente un indice de performance amélioré de 25% ($\Sigma.\Lambda = \sqrt{3}$) par rapport à l'opérateur optimal de Canny qui donne $\Sigma.\Lambda = 1{,}12$. Le cas correspondant à la limite de l'opérateur de Deriche pour ω tendant vers 0 présente le meilleur indice de performance (cas $m>>1$). Cette limite correspond à l'opérateur de dérivation h qui présente le meilleur compromis, donné par :

$$\boxed{h(x) = c.xe^{-\alpha|x|}} \quad \ldots\ldots(14) \quad ;$$

$$\text{Avec} \quad c = \frac{(1-e^{-\alpha})^2}{e^{-\alpha}}$$

Le filtre de lissage f s'obtient par intégration du filtre de dérivation :

$$\boxed{f(x) = \int_0^x h(x).dx = k(\alpha|x|+1)e^{-\alpha|x|}} \quad \ldots\ldots(15)$$

$$\text{Avec} \quad k = \frac{(1-e^{-\alpha})^2}{1+2\alpha.e^{-\alpha}-e^{-2\alpha}}$$

Grâce à la propriété de séparabilité, l'image lissée sera donnée par la convolution en 2D qui revient à effectuer deux convolutions en 1D en cascade :

$$I'(x,y) = I(x,y)*F(x,y) = I(x,y)*(f(x).f(y)) = (I(x,y)*f(x))*f(y) \quad \ldots\ldots(16)$$

Où $F(x,y)$ est la fonction de lissage en 2D définie par :

$$\boxed{F(x,y) = f(x).f(y)} \quad \ldots\ldots(17)$$

c- Filtre de Shen-Castan

Shen et Castan ont proposé un opérateur optimisant un critère incluant la détection et la localisation. Les critères qu'ils obtiennent correspondent aux critères de détection et de localisation de Canny et les filtres obtenus sont assez similaires dans la pratique. Le calcul du filtre de lissage optimal se fait par modélisation de la frontière par un échelon d'amplitude A noyé dans un bruit blanc stationnaire additif de moyenne nulle et de densité spectrale de puissance η_0^2. Le filtre de lissage obtenu par Shen et Castan s'écrit :

$$\boxed{f(x) = c.e^{-\alpha|x|}} \quad \ldots\ldots\ldots (18)$$

$$c = \frac{1-e^{-\alpha}}{1+e^{-\alpha}} \quad \text{(Pour avoir un filtre normalisé).}$$

Et le filtre de dérivation h correspondant :

$$\boxed{h(x) = \begin{cases} d.e^{-\alpha x} \ldots\ldots si \ldots x \geq 0 \\ d.e^{\alpha x} \ldots\ldots si \ldots x \leq 0 \end{cases}} \quad \ldots\ldots\ldots (19) \qquad \text{Avec } d = 1 - e^{-\alpha}$$

Le paramètre α définit la « largeur » du filtre ; plus α est petit, plus le lissage effectué par le filtre est important.

Remarquons que la discontinuité d'ordre 1 au point 0 du filtre de Shen permet d'éviter une délocalisation importante des contours dans l'image lissée, même avec des valeurs faibles de α. Cependant, cette discontinuité peut entraîner la détection de contours multiples.

I.7.5. Récursivité et implémentation des filtres

En utilisant l'opérateur optimal de Deriche, on veut mettre en oeuvre les techniques générales énoncées précédemment. Le filtrage récursif permet d'implémenter des filtres de réponse impulsionnelle infinie avec un coût algorithmique faible. On utilise donc cette technique pour programmer les filtres de lissage et de dérivation pour les opérateurs optimaux de Deriche et de Shen-Castan.

a- *Implémentation des filtres de Deriche*

Ce sont des filtres récursifs du second ordre dont les réalisations sont les suivantes :

- ***Filtre de lissage*** : $\boxed{l(x) = k(\alpha|x| + 1)e^{-\alpha|x|}}$ .....(20)

Si, par exemple, nous effectuons la convolution d'un signal mono-dimensionnel ***x(m)*** avec la fonction échantillonnée ***l(m)***, le signal final ***y(m)*** se déduit des équations suivantes :

$$y^+(m) = a_0 x(m) + a_1 x(m-1) + b_1 y^+(m-1) + b_2 y^+(m-2). \quad ....(21)$$

Pour m $1, ..., N$

$$y^-(m) = a_2 x(m+1) + a_3 x(m+2) - b_1 y^-(m+1) + b_2 y^-(m+2). \quad .....(22)$$

Pour $m-N, ..., 1$

$$y(m) = y^-(m) + y^+(m). \quad ..... \quad (23)$$

Pour $m-1, ..., N$

Où N représente la taille du signal et où les coefficients a_i, b_i et k se déduisent du paramètre α.

Tel que : $a_0 = k$, $a_1 = k(\alpha - 1)e^{-\alpha}$, $a_2 = k(\alpha + 1)e^{-\alpha}$,

$a_3 = -k.e^{-2\alpha}$,

$$b_1 = -2.e^{-\alpha}\ ,\ b_2 = e^{-2\alpha}\ ,\ \text{et}\quad k = \frac{(1-e^{-\alpha})^2}{1+2\alpha.e^{-\alpha}-e^{-2\alpha}}$$

- ***Filtre de dérivation*** : $\boxed{d(x) = -c.xe^{-\alpha|x|}}$ .....(24)

$$y^+(m) = ax(m) - b_1 y^+(m-1) - b_2 y^+(m-2). \quad .....(25)$$

Pour $m-1, ..., N.$

$$y^-(m) = -ax(m+1) - b_1 y^-(m+1) - b_2 y^-(m+2). \quad .....(26)$$

Pour $m \quad N, ..., 1$

$$y(m) = y^-(m) + y^+(m). \quad .....(27)$$

Pour $m-1, ..., N.$

Avec $a = c.e^{-\alpha}$, $b_1 = -2.e^{-\alpha}$, $b_2 = e^{-2\alpha}$ et $c = \dfrac{(1-e^{-\alpha})^2}{e^{-\alpha}}$

b- Implémentation des filtres de Shen

Les filtres de lissage et de dérivation de Shen s'implémentent par filtrage récursif d'ordre 1.

- ***Filtre de lissage :*** $\boxed{l(x) = c.e^{-\alpha|x|}}$ .....(28)

$$y^+(m) = ax(m) + by^+(m-1). \quad \text{Pour } m = 1,\ldots,N. \tag{29}$$

$$y^-(m) = ax(m+1) - by^-(m+1). \quad \text{Pour } m = N,\ldots,1. \tag{30}$$

$$y(m) = y^-(m) + y^+(m). \quad \text{Pour } m = 1,\ldots,N. \tag{31}$$

Avec $a = c = \dfrac{1-e^{-\alpha}}{1+e^{-\alpha}}$ et $b = e^{-\alpha}$

- *Filtre de dérivation :*

$$\boxed{h(x) = \begin{cases} d.e^{-\alpha x} \ldots\ldots si \ldots x \geq 0 \\ d.e^{\alpha x} \ldots\ldots si \ldots x \leq 0 \end{cases}} \ldots\ldots\ldots (32)$$

$$y^+(m) = dx(m) + fy^+(m-1). \quad \text{Pour } m = 1,\ldots,N. \tag{33}$$

$$y^-(m) = -dx(m+1) + fy^-(m+1). \quad \text{Pour } m = N,\ldots,1. \tag{34}$$

$$y(m) = y^-(m) + y^+(m). \quad \text{Pour } m = 1,\ldots,N. \tag{35}$$

Avec $d = 1 - e^{-\alpha}$ t et $f = e^{-\alpha}$

I.8. Conclusion

Dans ce chapitre, nous avons présenté des notions fondamentales ainsi que quelques techniques de traitement d'images, afin de préparer cette dernière à l'analyse. Nous avons présenté quelques traitements qui permettent une amélioration de l'image en vue d'éliminer toutes les imperfections qui existent sur cette dernière, dans le but de parvenir à une analyse plus fine.

II.1. Introduction

Le cerveau humain est capable de s'adapter, d'apprendre et de décider, et c'est sur ce fait que des chercheurs se sont intéressés à comprendre son principe de fonctionnement et de pouvoir l'appliquer au domaine de l'informatique. C'est ainsi que dans les années cinquante on formalisa le neurone en un modèle mathématique à partir du modèle biologique. [17]

Nous allons présenter dans ce chapitre une étude générale sur les réseaux de neurones, le passage du modèle biologique au modèle artificiel, le protocole d'apprentissage, ainsi les différentes classes et architectures de ces réseaux. Puis nous consacrerons une étude plus détaillée au réseau multicouche qui représente le modèle adéquat pour notre travail.

II.2. Modèle biologique.

II.2.1. Définition du neurone

Le neurone biologique est une cellule vivante spécialisée dans le traitement des signaux électriques.

Les neurones sont reliés entre eux par des liaisons appelées axones. Ces axones vont eux mêmes jouer un rôle important dans le comportement logique de l'ensemble. Ces axones conduisent les signaux électriques de la sortie d'un neurone vers l'entrée (synapse) d'un autre neurone.

Les neurones font une sommation des signaux reçus en entrée et en fonction du résultat obtenu, vont fournir un courant en sortie. [17]

II.2.2. Structure du neurone biologique

Le système nerveux compte plus de 1000 milliards de neurones interconnectés. Bien que les neurones ne soient pas tous identiques, leur forme et certaines caractéristiques permettent de les répartir en quelques grandes classes. En effet, les neurones n'ont pas tous un comportement similaire en fonction de leur position dans le cerveau. [17]

Voici le schéma d'un neurone biologique:

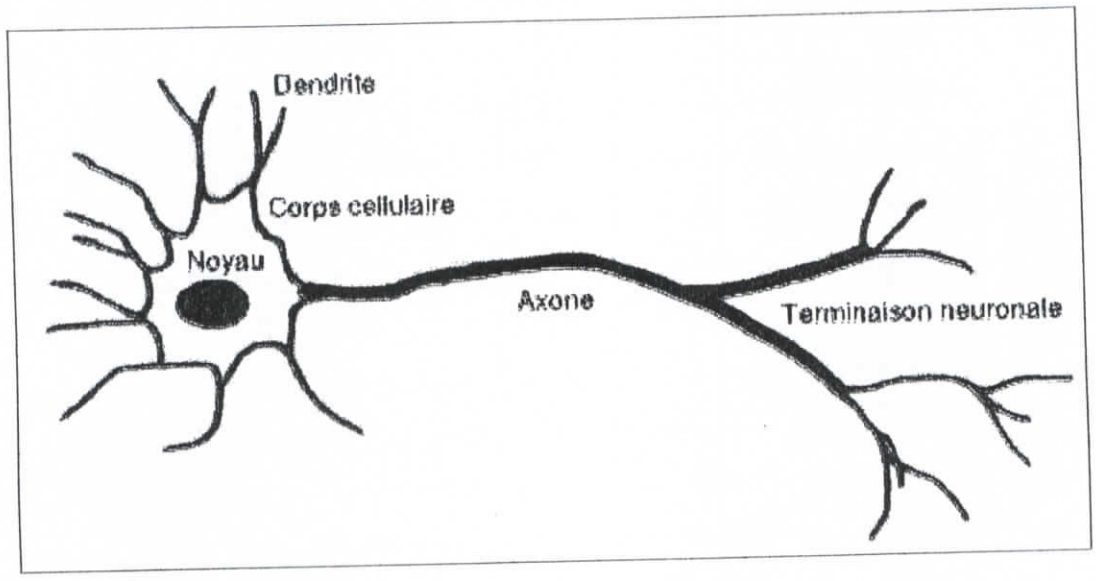


Figure II.1 : Structure du neurone biologique.

On peut le décomposer en trois régions principales:

Le corps cellulaire

Il contient le noyau du neurone ainsi que la machine biochimique nécessaire à la synthèse des enzymes. Ce corps cellulaire de forme sphérique ou pyramidale contient aussi les autres molécules essentielles à la vie de la cellule. Sa taille est de quelques microns de diamètre.

Les dendrites

Ce sont de fines extensions tubulaires qui se ramifient autour du neurone et forment une sorte de vaste arborescence. Les signaux envoyés au neurone sont captés par les dendrites. Leur taille est de quelques dizaines de microns de longueur.

L'axone

C'est le long de l'axone que les signaux partent du neurone. Contrairement aux dendrites qui se ramifient autour du neurone, l'axone est plus long et se ramifie à son extrémité ou il se connecte aux dendrites des autres neurones. Sa taille peut varier entre quelques millimètres à plusieurs mètres.

Synapse

Une synapse est une jonction entre deux neurones, et généralement entre l'axone d'un neurone et un dendrite d'un autre neurone (mais il existe aussi des synapses axo-axonales par exemple).

II.2.3. Fonctionnement

La membrane externe d'un neurone exécute cinq fonctions :

- Propagation des impulsions électriques le long de l'axone et des dendrites
- Libération des médiateurs à l'extrémité de l'axone.
- Réagir à ces médiateurs chimiques au niveau des dendrites.
- Réagir au niveau du corps cellulaire aux impulsions électriques issues des dendrites pour générer ou non une nouvelle impulsion.
- Permettre au neurone de reconnaître ses voisins pour se situer au cours de la formation du cerveau et trouver à quelles autres cellules il doit ce connecter.

II.3. Modèle artificiel

II.3.1. Définition du réseau de neurones artificiel

Ce que l'on désigne habituellement par "**réseau de neurones**" (neural network), ou réseau neuromimétique est un réseau de neurones artificiels basé sur un modèle simplifié de neurone. Ce modèle permet certaines fonctions du cerveau, comme la mémorisation associative, l'apprentissage par l'exemple, le travail en parallèle, etc. Cependant le neurone formel ne possède pas toutes les capacités des neurones biologiques, comme le partage de synapses, l'activation membranaire ou la structuration prénatale des neurones, par conséquent les réseaux de neurones actuels sont loin d'avoir les possibilités du cerveau. [17]

II.3.2. Fonctionnement du réseau de neurones

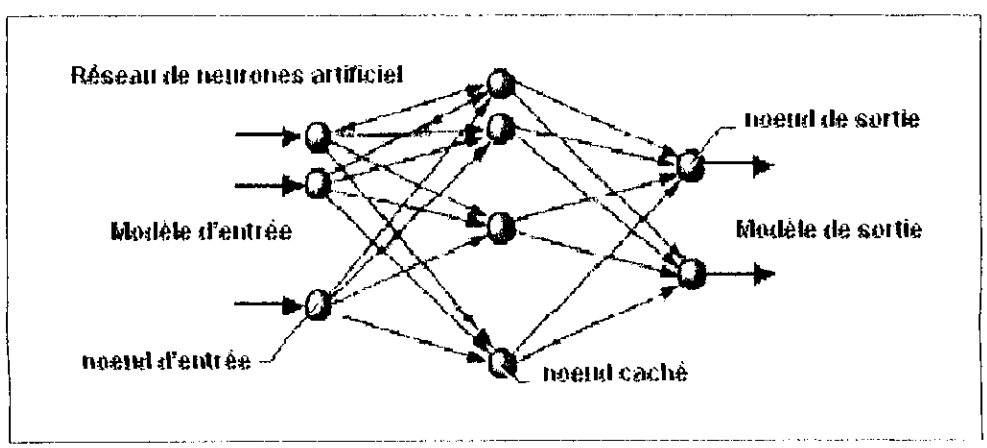


Figure II.2 : Réseau de neurones artificiel.

Le fonctionnement des systèmes de neurones se caractérise par trois principes :

- Le parallélisme des opérations de traitement.
- Le caractère collectif et la distribution de l'activité des éléments du réseau.
- Les capacités d'apprentissage à partir d'exemple.

Les simulations des systèmes de neurones ont manifesté des comportements analogues à ceux du modèle vivant : mémoire à long terme et à court terme. Il est même observé qu'il existe une limite du nombre d'article enregistrable en une seule fois ; si cette limite est dépassée, des objets de la mémoire sont effacés : c'est ce que l'on appelle l'effet " PALIMPSESTE ".

On pourrait penser que plus le nombre de neurones est important, plus le réseau est performant. Ceci n'est pas certain. Lorsque les concepteurs des réseaux essayent d'augmenter le nombre de neurones, ils sont limités par plusieurs contraintes, et en particulier par la durée d'apprentissage qui augmente avec le nombre de neurones. En effet, l'apprentissage s'effectue par l'exemple et il faut que le nombre de ceux-ci soit bien plus important que le nombre de connexions. Quand on sait qu'un système de 1000 neurones possède plus de 1000 000 connexions, on peut craindre le temps d'apprentissage. [17]

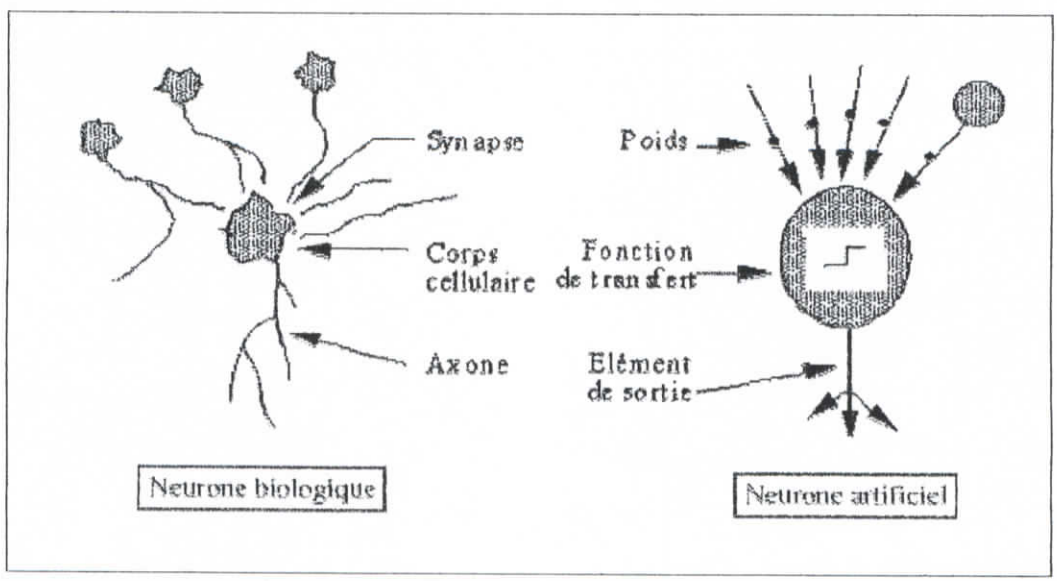


Figure II.3 : Dualité entre un neurone artificiel et biologique.

II.3.3. Neurone formel

Le premier neurone formel est apparu en 1943. On le doit à Mac Culloch et Pitts. Ils ont prouvé que pour des poids judicieusement choisis ce modèle offrait la puissance d'une machine de Turing universelle.

Voici un schéma de leur modèle de neurone formel :

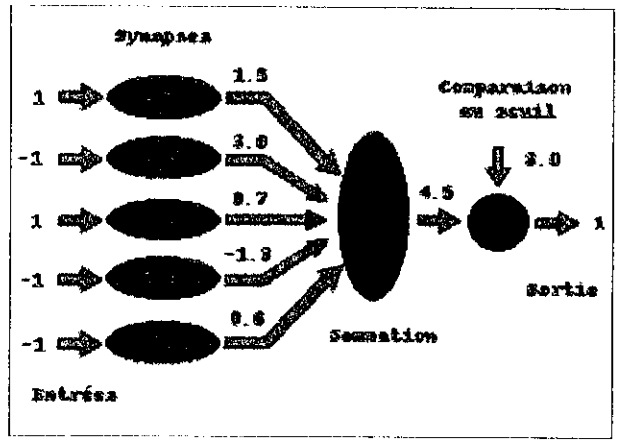


Figure II.4 : Schéma du neurone formel (Mac Culloch et Pitts).

Le neurone formel est donc une modélisation mathématique qui reprend les principes du fonctionnement du neurone biologique, en particulier la sommation des entrées. Sachant qu'au niveau biologique, les synapses n'ont pas toutes la même «valeur» (les connexions entre les neurones étant plus ou moins fortes), les auteurs ont donc créé un algorithme qui pondère la somme de ses entrées par des poids synaptiques (coefficients de pondération). Les 1 et les -1 en entrée sont là pour figurer une synapse excitatrice ou inhibitrice.

Interprétation mathématique

D'un point de vue mathématique, le neurone formel peut être représenté de la manière suivante:

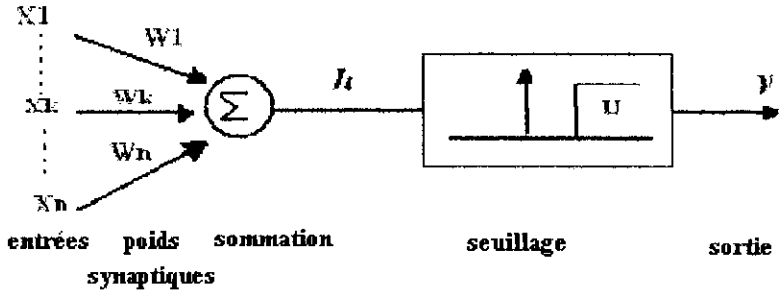


Figure II.5 : Représentation mathématique du neurone formel.

Pour un nombre compris entre j (=1) et un nombre quelconque n, le neurone formel va calculer la somme de ses entrées (x_1, ..., x_n), pondérées par les poids synaptiques (w_1, ..., w_n), et la comparer à son seuil θ. Si le résultat est supérieur au seuil, alors la valeur renvoyée est 1, sinon la valeur renvoyée est 0.

D'où la formule:

$$y = f\left(\sum_{j=1}^{n}(w_j x_j - \theta)\right) \quad \text{.....(36)}$$

Où f : fonction de seuil.

La sortie se calcule de la manière suivante:

```
Out (Entrée du neurone X[])
{       Sortie=0;
        Pour tout poids W[i] faire
                Sortie+=W[i]*X[i];
        Fin Pour
        Sortie=H (Sortie);
        Retourner Sortie;
}
```

II.3.4. Fonctions d'activation

Chaque neurone calcule sa valeur de sortie y à partir de la somme pondérée de ses entrées et de ses poids, il existe différentes fonctions d'activation permettant de calculer cette valeur.

Dans sa première version, le neurone formel était donc implémenté avec une fonction à seuil, mais de nombreuses versions existent. Ainsi le neurone de McCulloch et Pitts a été généralisé de différentes manières, en choisissant d'autres fonctions d'activations, comme les fonctions linéaires ou les sigmoïdes par exemple.

Voici un tableau récapitulatif de différents types de fonctions d'activation les plus utilisées, avec leurs équations mathématiques et leurs dérivées. [17]

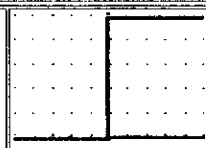
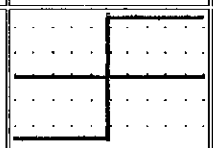
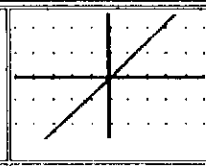
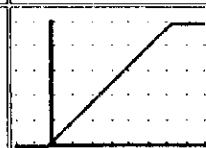
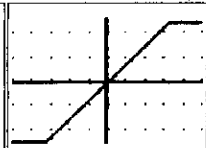
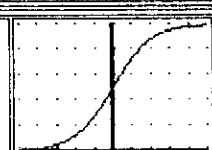
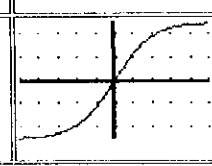
Catégorie	Type	Equation	Allure	Dérivée
Seuil	**Binaire (fonction de Heaviside)**	$f(x) = 1$ si $x > 0$ $f(x) = 0$ si $x \leq 0$		-
	Signe	$f(x) = 1$ si $x > 0$ $f(x) = -1$ si $x \leq 0$		-
Linéaire	**Identité**	$f(x) = x$		$f'(x) = 1$
	Saturé positif	$f(x,k) = 0$ si $x < 0$ $f(x,k) = 1$ si $x \geq 1/k$ $f(x,k) = k.x$ sinon		$f'(x,k) = 0$ si $x < 0$ $f'(x,k) = 0$ si $x \geq 1/k$ $f'(x,k) = k$ sinon
	Saturé symétrique	$f(x,k) = -1$ si $x < -1/k$ $f(x,k) = 1$ si $x \geq 1/k$ $f(x,k) = k.x$ sinon		$f'(x,k) = 0$ si $x < -1/k$ $f'(x,k) = 0$ si $x \geq 1/k$ $f'(x,k) = k$ sinon
Sigmoïde	**Positive (type logistique)**	$f(x,k) = \frac{1}{1+e^{-k.x}}$		$f'(x,k) = \frac{k}{2+e^{-k.x}+e^{k.x}}$
	Symétrique (type tanh)	$f(x,k) = \frac{1}{1+e^{-k.x}} - 1$		$f'(x,k) = \frac{2k}{2+e^{-k.x}+e^{k.x}}$

Tableau II.1 : Tableau récapitulatif de différents types de fonctions d'activation.

II.4. Protocole d'apprentissage des réseaux de neurones

II.4.1. Procédure d'apprentissage

L'apprentissage d'un réseau se fait généralement dans le contexte d'une tache ou d'un comportement à apprendre. Les informations à traiter sont codées sous la forme d'un vecteur appelé *patron d'entrée*, qui est communiqué aux neurones d'entrée du réseau. La réponse du réseau s'interprète à partir de la valeur d'activation de ses neurones de sorties, dont le vecteur s'appelle *patron de sortie*. Lors d'un apprentissage supervisé, on dispose aussi du comportement de référence que doit apprendre le réseau, exprimé sous la forme de *patron de référence*, ou *patron de sorties désirées*.
En général, l'apprentissage se fait sur une période relativement longue, durant laquelle les patrons d'entrée (et éventuellement de sorties désirées) peuvent être présentés au réseau un grand nombre de fois chacun. Cet apprentissage comprend quatre étapes de calcul :

1. *Initialisation des poids synaptiques du réseau*. En général, les poids trouvés par le réseau à la fin de l'apprentissage dépendent en partie de l'ensemble des poids dont il disposait au départ. Sauf cas exceptionnel, le choix exact de ces valeurs initiales n'est pas critique pour le succès de l'apprentissage, et la pratique courante est d'initialiser les poids du réseau à des (petites) valeurs aléatoires au début de l'apprentissage.
2. *Présentation du patron d'entrée et propagation d'activation*.
3. *Calcul de l'erreur*. Pour chaque neurone du réseau, une valeur *d'erreur* est calculée à partir de son activation et de celui des neurones qui lui sont reliés. Dans le cas d'un apprentissage supervisé, l'erreur tient aussi compte de la différence entre l'activation des neurones de sortie et du patron de référence.
4. *Calcul du vecteur de correction*. A partir des valeurs d'erreur, on détermine alors la correction à apporter aux poids synaptiques des connexions et aux seuils des neurones. La correction effective des poids peut se faire après chaque présentation de patron ; alternativement, les vecteurs de correction peuvent être accumulés pendant un certain temps avant d'être appliqués au réseau. Le

nombre de patrons à présenter au réseau avant d'effectuer la correction s'appelle *fenêtre de mise à jour* (batch size).

Les étapes 2-3-4 sont répétées jusqu'à la fin de l'apprentissage. [17]

II.4.2. Procédure de validation croisée

La validation croisée procède en trois étapes : [17]

- *L'apprentissage même*, pendant lequel on évalue le comportement du réseau sur le corpus d'apprentissage.
- *Une étape de test*, où l'apprentissage est suspendu périodiquement, et où le corpus test est présenté au réseau. Les performances ainsi obtenues constituent une première indication des performances réelles du réseau.
- *Une étape de validation a posteriori*. Un troisième corpus (dit de validation) est présenté au réseau à la fin de l'apprentissage. Les performances du réseau sur celui-ci sont employées pour juger du succès ou de l'échec de l'apprentissage.

II.4.3. Types d'apprentissage

Les procédures d'apprentissage peuvent être classées en trois catégories : non supervisé, supervisé et semi-supervisé. [21]

II.4.3.1. Apprentissage non supervisé

L'apprentissage non supervisé, dans lequel on se contente de présenter des formes sans indication sur leur identité. Une règle d'apprentissage est la règle de Hebb (voir annexe 2) qui revient à augmenter le poids de la connexion entre deux cellules si celles-ci sont simultanément actives et à le diminuer dans le cas contraire. Les cartes de Kohonen (qu'on verra par la suite), utilisent un apprentissage non supervisé compétitif, inspiré de la loi de Hebb. Un autre exemple de modèle à apprentissage non supervisé est le modèle ART.

II.4.3.2. Apprentissage supervisé

Dans l'apprentissage supervisé, on impose une réponse en sortie du réseau pour une forme d'entrée donnée (dispose d'un comportement de référence vers lequel il tente de faire converger le réseau). Dans ce cas, il s'agit de calculer l'erreur commise

par le réseau comme une fonction de la "distance" entre la sortie désirée et la sortie obtenue. Il existe deux grands types d'erreurs utilisés en pratique, correspondant à deux critères d'optimisation :

- le critère des moindres carrés fondé sur l'utilisation d'une distance euclidienne entre les formes,
- le critère d'entropie croisée dans lequel on considère les sorties du réseau comme des distributions de probabilités sur les variables d'entrée. Il s'agit alors de maximiser l'information mutuelle entre les sorties réelles et désirées.

Dans le cas du perceptron multicouches, l'algorithme d'apprentissage, dit de rétropropagation du gradient d'erreur, est itératif (voir annexe 2). Le principe est d'adapter les différents poids des connexions du réseau à chaque présentation d'une forme en entrée, selon le gradient de l'erreur commise en sortie.

II.4.3.3. Apprentissage semi-supervisé

L'apprentissage semi-supervisé suppose qu'un comportement de référence précis n'est pas disponible, mais qu'en revanche il est possible d'obtenir des indications qualitatives (par exemple, correct/incorrect) ou lacunaires (par exemple, une indication tous les N patrons) sur les performances du réseau.

II.5. Architecture et classification des réseaux de neurones

On peut classer les RNA en deux grandes catégories: [17]

II.5.1. Les réseaux non bouclés "FEED-FORWARD"

Appelés aussi "réseaux de type Perceptron", ce sont des réseaux dans lesquels l'information se propage de couche en couche sans retour en arrière possible.
Ce type de réseau réalise une fonction algébrique de ses entrées par composition des fonctions réalisées par chacun des neurones.
Ce type de réseau est dit acyclique car si l'on se déplace dans un réseau à partir d'un neurone quelconque suivant les connexions, on ne peut pas revenir au neurone de départ.

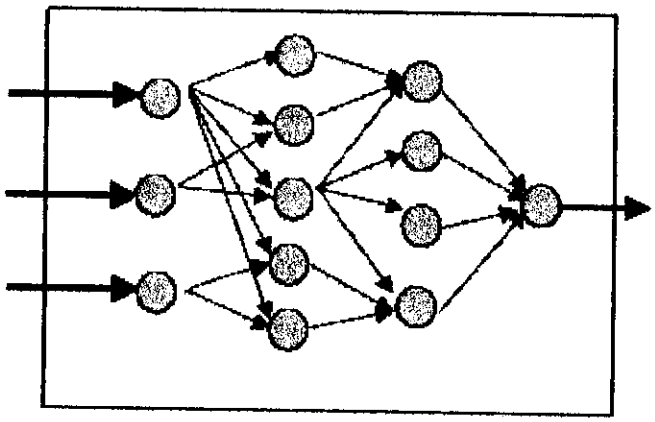


Figure II.6 : Réseau de neurones non bouclé.

II.5.1.1. Les Perceptrons

a- Le Perceptron monocouche :

C'est historiquement le premier RNA, c'est le Perceptron de Rosenblatt. C'est un réseau simple, puisque il ne se compose que d'une couche d'entrée et d'une couche de sortie. Il est calqué, à la base, sur le système visuel et de ce fait a été conçu dans un but premier de reconnaissance des formes. Cependant, il peut aussi être utilisé pour faire de la classification et pour résoudre des opérations logiques simples (telle "ET" ou "OU"). Sa principale limite est qu'il ne peut résoudre que des problèmes linéairement séparables. Il suit généralement un apprentissage supervisé selon la règle de correction de l'erreur (ou selon la règle de Hebb).

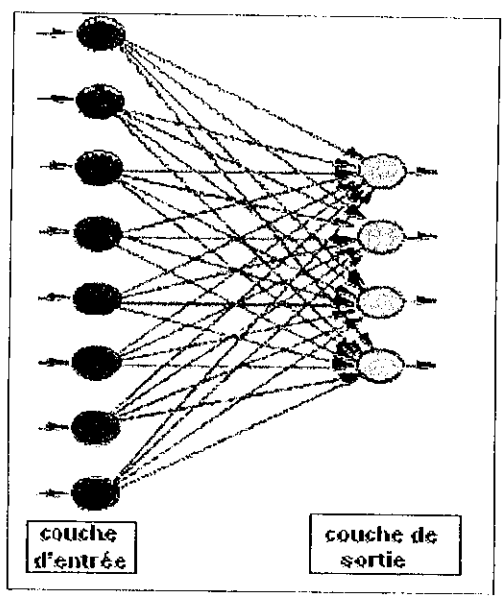


Figure II.7 : Perceptron monocouche.

b- Le Perceptron multicouche (PMC)

C'est une extension du précédent, avec une ou plusieurs couches cachées entre l'entrée et la sortie. Chaque neurone dans une couche est connecté à tous les neurones de la couche précédente et de la couche suivante (excepté pour les couches d'entrée et de sortie) et il n'y a pas de connexions entre les cellules d'une même couche. Les fonctions d'activation utilisées dans ce type de réseaux sont principalement les fonctions à seuil ou sigmoïdes. Il peut résoudre des problèmes non linéairement séparables et des problèmes logiques plus compliqués, et notamment le fameux problème du XOR. Il suit aussi un apprentissage supervisé selon la règle de correction de l'erreur.

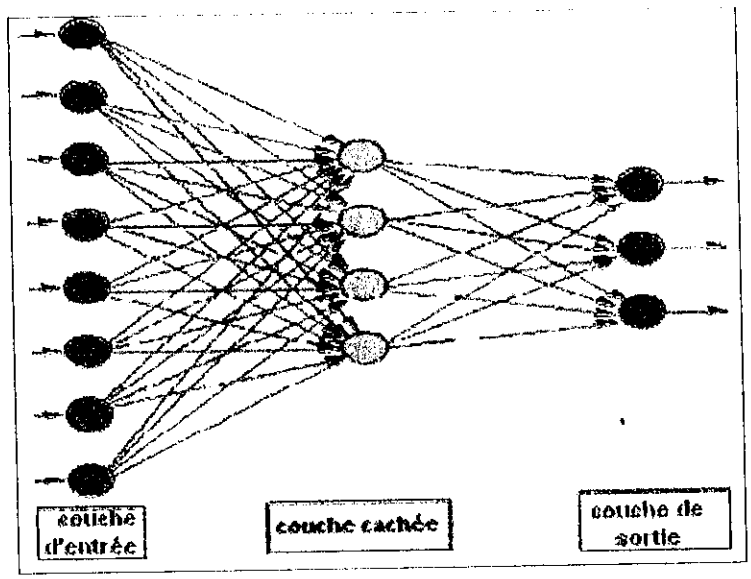


Figure II.8 : Perceptron multicouche (à trois couches).

II.5.1.2. Les réseaux à fonction radiale

Ce sont les réseaux que l'on nomme aussi RBF ("Radial Basic Functions"). L'architecture est la même que pour les PMC cependant, les fonctions de base utilisées ici sont des fonctions Gaussiennes. Les RBF seront donc employés dans les mêmes types de problèmes que les PMC à savoir, en classification et en approximation de fonctions, particulièrement. L'apprentissage le plus utilisé pour les RBF est le mode hybride et les règles sont soit, la règle de correction de l'erreur soit, la règle d'apprentissage par compétition.

II.5.2. Les réseaux bouclés "FEED-BACK"

Appelés aussi "réseaux récurrents", ce sont des réseaux dans lesquels il y a retour en arrière de l'information.

C'est l'architecture la plus générale pour un réseau de neurones et la plus répandue ; les réseaux bouclés possèdent un graphe cyclique tel que lorsqu'on se déplace dans le réseau suivant le sens des connexions, on peut toujours revenir vers notre point de départ.

La sortie d'un neurone du réseau peut donc être fonction d'elle-même ; la notion de retard est donc introduite.

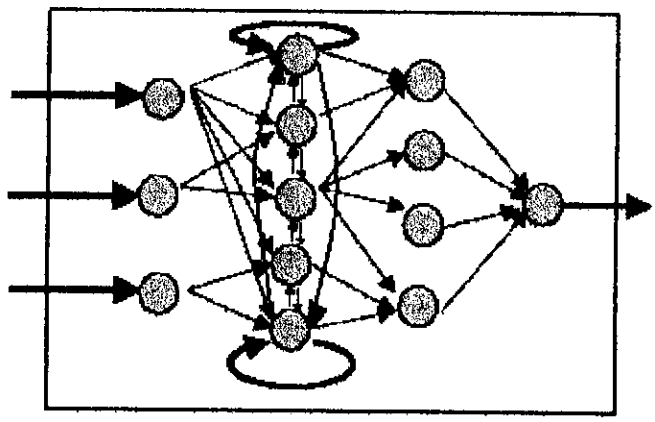


Figure II.9 : Réseau de neurones bouclé.

II.5.2.1. Les cartes auto-organisatrices de Kohonen

Ce sont des réseaux à apprentissage non-supervisé qui établissent une carte discrète, ordonnée topologiquement, en fonction de patterns d'entrée. Le réseau forme ainsi une sorte de treillis dont chaque noeud est un neurone associé à un vecteur de poids. La correspondance entre chaque vecteur de poids est calculée pour chaque entrée. Par la suite, le vecteur de poids ayant la meilleure corrélation, ainsi que certains de ses voisins, vont être modifiés afin d'augmenter encore cette corrélation.

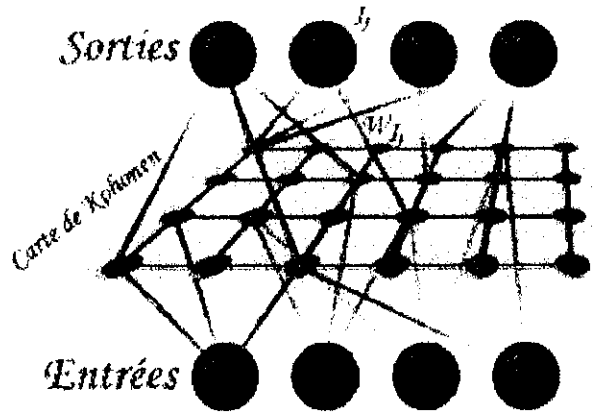


Figure II.10 : Carte auto-organisatrice de Kohonen.

II.5.2.2. Les réseaux de Hopfield

Les réseaux de Hopfield sont des réseaux récurrents et entièrement connectés. Dans ce type de réseau, chaque neurone est connecté à chaque autre neurone et il n'y a aucune différenciation entre les neurones d'entrée et de sortie. Ils fonctionnent comme une mémoire associative non-linéaire et sont capables de trouver un objet stocké en fonction de représentations partielles ou bruitées. L'application principale des réseaux de Hopfield est l'entrepôt de connaissances mais aussi la résolution de problèmes d'optimisation. Le mode d'apprentissage utilisé ici est le mode non-supervisé.

II.5.2.3. Les ARTs

les réseaux ARTs ("Adaptative Resonnance Theorie") sont des réseaux à apprentissage par compétition. Le problème majeur qui se pose dans ce type de réseaux est le dilemme « stabilité/plasticité ». En effet, dans un apprentissage par compétition, rien ne garantit que les catégories formées vont rester stables. La seule possibilité, pour assurer la stabilité, serait que le coefficient d'apprentissage tende vers zéro, mais le réseau perdrait alors sa plasticité. Les ART ont été conçus spécifiquement pour contourner ce problème. Dans ce genre de réseau, les vecteurs de poids ne seront adaptés que si l'entrée fournie est suffisamment proche, d'un prototype déjà connu par le réseau. On parlera alors de résonance. A l'inverse, si l'entrée s'éloigne trop des

prototypes existants, une nouvelle catégorie va alors se créer, avec pour prototype, l'entrée qui a engendrée sa création. Il est à noter qu'il existe deux principaux types de réseaux ART : les ART-1 pour des entrées binaires et les ART-2 pour des entrées continues. Le mode d'apprentissage des ARTs peut être supervisé ou non.

Récapitulation

Les différents réseaux de neurones peuvent être classés comme dans le schéma ci dessous:

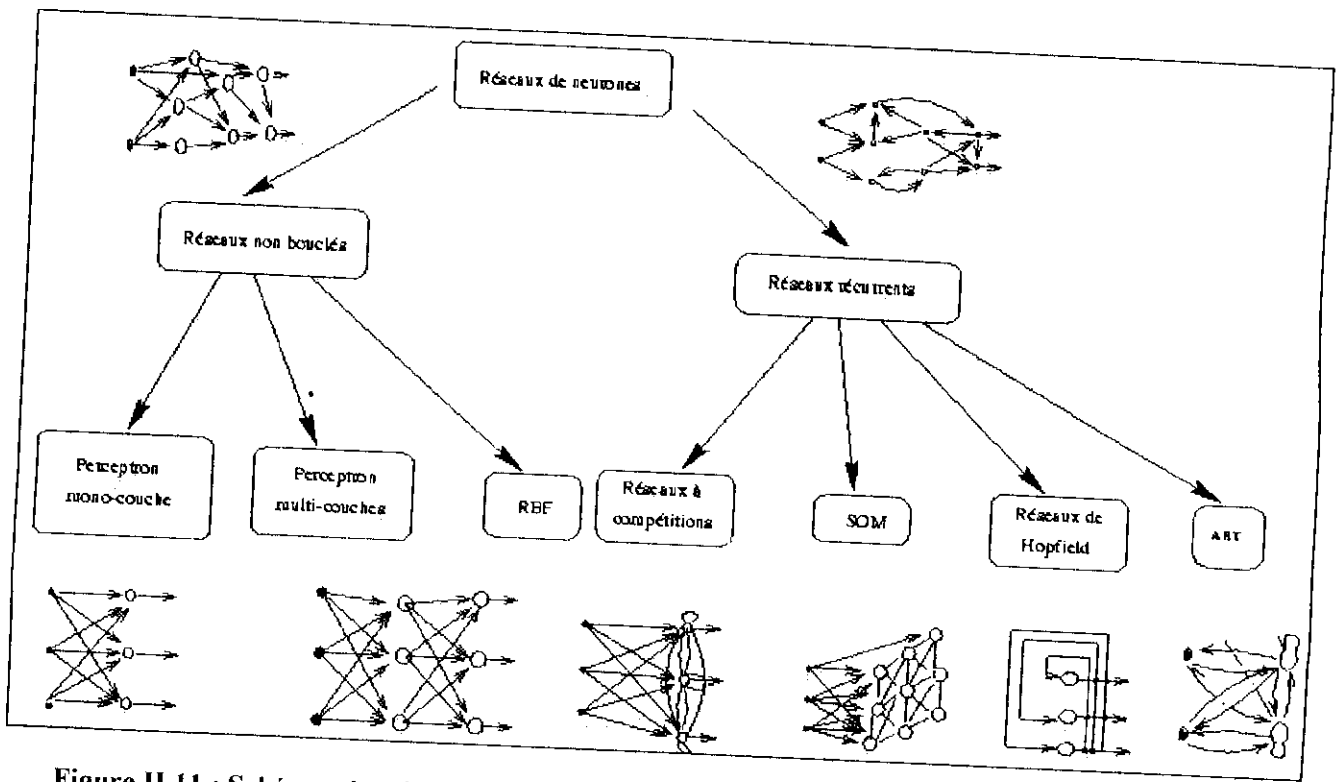


Figure II.11 : Schéma récapitulatif des différentes classes des réseaux de neurones.

II.6. Comparaison

Les avantages appréciables caractérisant les réseaux de neurones (rapidité, robustesse, adaptabilité...etc.), ont incité les scientifiques à une large exploitation. [18] En effet, le choix du type de réseau et de son genre d'implémentation se base sur :

- La nature de l'application
- La nature des données
- Les considérations sur les performances

Le tableau suivant donne une sélection de deux types de réseaux pour différentes applications.

Modèle / Application	Réseaux multicouches	Réseaux de Kohonen
Classification	✓	✓
Reconnaissance des formes	✓	✓
Aide à la décision	✓	✓
Optimisation		✓

Tableau II.2 : comparaison entre les réseaux multicouches et les réseaux de Kohonen

II.7. Domaines d'applications des réseaux de neurones

Voici quelques exemples d'applications : [17]

- *La reconnaissance de forme.* Dans ce type de problème, les données d'entrée représentent l'information recueillie par un ou plusieurs capteurs (caméra, sonar, micro). Le but est de reconnaître le ou les objets perçus par le capteur.

  - *En reconnaissance de la parole :* Les entrées peuvent être une segmentation de la fréquence d'un signal sonore, et les sorties identifient la valeur de tel ou tel phénomène associé à l'entrée. Ou bien on peut transcrire le langage parlé en texte ASCII.
  - *En reconnaissance de cibles :* Applications militaires qui utilisent des données issues d'images vidéo ou infrarouge pour déterminer la présence d'une cible ennemie.
  - *En reconnaissance de Codes postaux :* Cette application fût l'un des premiers problèmes de reconnaissance d'images assez complexe, en raison de la variabilité des styles d'écritures, mais en plus cette application est moins onéreuse que de rectifier une erreur de tri.

  - *En reconnaissance d'écriture :* Qu'elle soit manuscrite ou imprimée, le réseau va être confronté à un problème de classification, par rapport à la base d'apprentissage qui constitue la base de données.

- *La classification.* Ce type d'application ressemble beaucoup à la reconnaissance de formes, en ce que son but est d'associer à une entrée donnée l'étiquette d'une forme connue.

- *La transformation.* Plusieurs applications ont été proposées où le réseau doit apprendre une transformation mathématique entre l'entrée et la sortie. En traitement d'image, par exemple, il existe des modèles connexionnistes capables de calculer les contours d'objets, la correspondance temporelle d'objets en mouvement et la compression de données.

- *La prédiction et le contrôle de processus.* Le problème de la prédiction consiste à estimer l'état futur d'un processus à partir d'un historique de son comportement et des variables environnementales attenantes. En contrôle, il s'agit non seulement d'estimer passivement l'état futur, mais aussi d'intervenir de façon à atteindre un but donné. Dans ce cadre, un domaine où les réseaux ont un succès incontestable est dans le domaine de la finance. Dans un contexte plus industriel, les applications sont diverses ; citons la prédiction de séries temporelles (par exemple, la consommation électrique ou le contrôle automatique de processus dynamiques.

- *En médecine.* Les entrées du réseau peuvent être un ensemble de valeurs résultant d'une liste de tests médicaux, les sorties étant les symptômes possibles de maladies infectieuses.

II.8. Propriétés et limites des réseaux de neurones

II.8.1. Propriétés

L'intérêt porté aujourd'hui aux réseaux de neurones tient sa justification dans les quelques propriétés fascinantes qu'ils possèdent : [17]

- ***La capacité d'adaptation.*** Elle se manifeste par la capacité d'apprentissage qui permet de tenir compte de nouvelles contraintes ou de nouvelles données du monde extérieur.

- ***Le parallélisme.*** Les réseaux de neurones sont considérés comme un ensemble d'entités élémentaires qui travaillent simultanément. Le parallélisme permet une rapidité de calcul supérieur mais exige de poser différemment les problèmes à résoudre.
- ***La capacité de généralisation.*** La capacité de généralisation d'un réseau de neurones est sa capacité à donner une réponse satisfaisante à une entrée qui ne fait partie de son ensemble d'apprentissage.

II.8.2. Limites

Les principales limites actuelles sont : [17]

- La plupart des réseaux de neurones sont simulés sur des machines séquentielles. Ce qui entraîne des temps de calculs importants dés que la taille du problème devient grande.
- Incapacité d'expliquer les résultats qu'ils fournissent. Les réseaux de neurones se comportent comme des boites noires dont les règles de fonctionnement sont inconnues.

II.9. Réseau multicouche

Apparus en 1985, les réseaux multicouches sont aujourd'hui les modèles les plus employés, ils sont intemporels (réseaux statiques et non dynamiques).
Plusieurs couches de traitement leurs permettent de réaliser des associations non linéaires entre l'entrée et la sortie. Ils sont ainsi capables de résoudre le cas du "ou exclusif". Depuis les années soixante que les possibilités de traitement des réseaux multicouches sont supérieures à celle du Perceptron, cependant l'algorithme d'apprentissage manquait. Pour la couche de sortie, on peut appliquer l'apprentissage du Perceptron, mais comment modifier les poids pour les connexions qui ne sont pas en relation avec un neurone de sortie ?
Le problème est ramené à l'obtention d'une estimation de la valeur désirée pour chaque neurone de la couche cachée. La rétropropagation (backpropagation) de gradient est une solution à ce problème (voir annexe 2).
Le principe utilisé par la rétropropagation de gradient est la minimisation d'une fonction dépendante de l'erreur.

II.9.1. Structure et fonctionnement

Les neurones sont organisés en couches, chaque neurone est connecté à toutes les sorties des neurones de la couche précédente, et nourrit de sa sortie tous les neurones de la couche suivante (ces réseaux sont d'ailleurs qualifiés de *feedforward*, "nourrit devant"). Pour la première couche ses entrées sont l'entrée du réseau. D'ailleurs une couche est souvent rajoutée pour constituer les entrées, appelée couche d'entrée, mais elle n'en est pas une puisqu'elle ne réalise aucun traitement.

Les fonctions d'entrée et de transfert sont les mêmes pour les neurones d'une même couche, mais peuvent différer selon la couche. Ainsi la fonction de transfert de la couche de sortie est généralement l'identité. [17]

On se retrouve facilement avec des choses ressemblant à ceci :

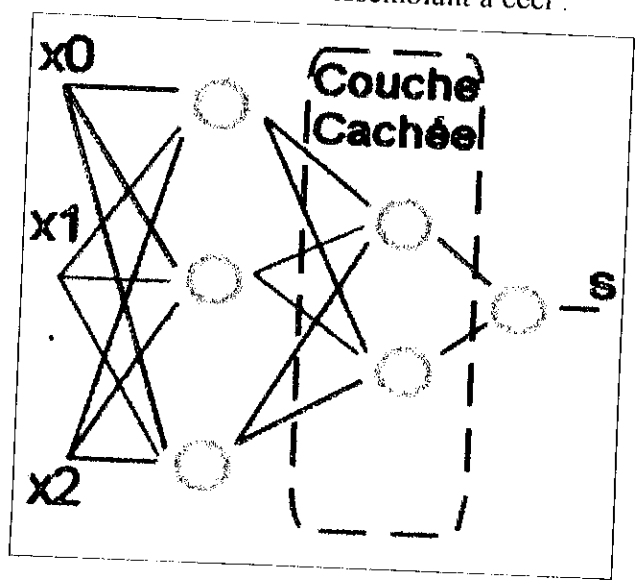


Figure II.12 : Perceptron multicouche.

Les cercles représentent les neurones, les traits précédant les neurones correspondent aux poids des neurones (les couleurs ne sont là que pour aider à la lisibilité et n'ont aucune importance). Les Xi sont la rétine (les paramètres d'entrée) et S est l'ensemble de décision (ou sortie). Ce Perceptron multicouche se désigne ainsi : {3,2,1}. 3 neurones en couche 0, 2 en couche 1 et 1 en couche 2.

Ceci ne constitue qu'un exemple de PMC (Perceptron multicouche). Ainsi, il est courant de trouver des PMC de ce type : {200,100,50} ou autre. Il est également possible d'avoir plusieurs couches cachées : {4,3,2,5,2,1,4}. Les PMC sont véritablement des boîtes noires chaotiques, il est d'ailleurs impossible de comprendre pourquoi les poids, une fois le réseau apprit, sont ainsi.

Les PMC ont deux fonctions principales : Apprendre (Learn) et Sortie (Out).

II.9.2. Calcul de la fonction sortie (Out)

On va tout simplement calculer successivement les sorties de chaque couche et les présenter en entrée de la couche suivante. Voyons donc tout d'abord une fonction permettant à une couche de retourner sa sortie :

L'algorithme est le suivant :

```
Out (Entrée de la couche X[])
{
        Pour chaque Neurone N[i] de la couche d'entrée faire
                S[i]=N[i].out(X);
        Fin Pour
        Retourner S;
}
```

Une fois que nous avons cette fonction, il ne nous reste plus qu'à calculer la succession des sorties :

```
Out (Rétine X[])
{
        Pour chaque Couche C[i] du PMC faire
                X=C[i].out(X);
        Fin Pour
        Retourner X;
}
```

Après cela, X contient la sortie du PMC. Cependant, il fait encore bien des erreurs, normal : il n'a rien apprit. Il nous reste donc à voir la fonction d'apprentissage du PMC.

II.9.3. Apprentissage par rétropropagation du gradient (Learn)

Les réseaux multicouches utilisent la règle de rétropropagation du gradient (décrite en annexe 2). Les fonctions de transferts doivent donc être différentiables, c'est pourquoi on utilise des fonctions sigmoïdes, qui sont une approximation infiniment dérivable de

la fonction à seuil de Heaviside.

Le principe est simple, il s'agit de calculer la sortie du PMC suivant une série d'échantillons, et de calculer l'erreur en comparant la sortie du PMC avec la sortie attendue, l'erreur étant ensuite corrigée. [17]

L'algorithme est le suivant :

```
Learn(Rétine X[],Sortie Attendue A[])
{
        //Calcul de la sortie
        S=Out(X);
        //Calcul des erreurs Dab (a étant la couche et b le neurone) par rétropropagation
        Pour tout Neurone i de la dernière couche j faire
                Dji=Si*(1-Si)*(A[i]-Si);
        Fin Pour
        Pour chaque couche i de l'avant dernière couche à la première faire
                //Si = Sortie de la couche i
                Pour chaque neurone j de la couche i faire
                        Dij=0;
                        //Sij=Sortie du neurone j de la couche i
                        Pour chaque neurone k de la couche l=i+1 faire
                                //Wlkj = Poids du neurone k associé à l'entrée en
provenance du neurone j
                                Dij+=Dlk*Wlkj;
                        Fin Pour
                        Dij*=Sij*(1-Sij);
                Fin Pour
        Fin Pour
        //Mise à jour des poids
        Pour tout poids Wijk (poids du neurone j de la couche i associé à l'entrée
provenant du neurone k de la couche i-1) faire
                //Sk = Sortie du neurone k
                Wijk+=Dij*Sk
        Fin Pour
}
```

A présent l'erreur se corrige. Le réseau neuronal est capable d'apprendre de nouvelles informations en temps réel.

II.9.4. Mise en oeuvre d'un réseau neuronal

La mise en œuvre d'un réseau neuronal est composée de quatre étapes principales : [21]

➢ Étape 1 : fixer le nombre de couches cachées

Mis à part les couches d'entrée et de sortie, l'analyste doit décider du nombre de couches intermédiaires ou cachées. Sans couche cachée, le réseau n'offre que de faibles possibilités d'adaptation ; avec une couche cachée, il est capable, avec un nombre suffisant de neurones, d'approximer toute fonction continue. Une seconde couche cachée prend en compte les discontinuités éventuelles.

➢ Étape 2 : déterminer le nombre de neurones par couches cachées

Chaque neurone supplémentaire permet de prendre en compte des profils spécifiques des neurones d'entrée. Un nombre plus important permet donc de mieux coller aux données présentées mais diminue la capacité de généralisation du réseau. Dans ce cas non plus il n'existe pas de règle générale mais des règles empiriques. La taille de la couche cachée doit être :

- soit égale à celle de la couche d'entrée,
- soit égale à 75% de celle-ci,
- soit égale à la racine carrée du produit du nombre de neurones dans la couche d'entrée et de sortie.

Notons que le dernier choix réduit le nombre de degrés de liberté laissés au réseau, et donc la capacité d'adaptation sur l'échantillon d'apprentissage, au profit d'une plus grande stabilité/capacité de généralisation.

Une voie de recherche ultérieure consisterait soit à procéder à l'estimation d'un réseau comportant de nombreux neurones puis à le simplifier par l'analyse des multicolinéarités ou par une règle d'apprentissage éliminant les neurones inutiles ; soit à définir une architecture tenant compte de la structure des variables identifiée au préalable par une analyse en composantes principales.

➢ Étape 3 : choisir la fonction d'activation

Nous considérerons la fonction logistique pour le passage de la couche d'entrée à la couche cachée. Le passage de cette dernière à la couche de sortie sera soit linéaire, soit logistique selon nos types de variables.

➢ Étape 4 : choisir l'apprentissage

L'apprentissage de rétropropagation nécessite la détermination du paramètre d'ajustement des poids synaptiques à chaque itération.
La détermination du critère d'arrêt est aussi cruciale dans la mesure où la convergence peut passer par des minima locaux.

II.10. Conclusion

À travers ce chapitre, nous constatons que les réseaux de neurones sont une alternative qui peut être très efficace pour les problèmes que les algorithmes classiques ne peuvent résoudre. Leur capacité d'apprentissage automatique, permet de résoudre des problèmes sans nécessiter l'écriture de règles complexes, tout en étant tolérant aux erreurs. Cependant, ce sont de véritables boites noires qui ne permettent pas d'interpréter les modèles construits. En cas, d'erreurs du système, il est quasiment impossible d'en déterminer la cause.

Enfin, il faut mentionner que le développement d'un système à base de réseaux neuromimétiques est une tâche délicate et qui nécessite beaucoup d'expérience. De nombreux problèmes se posent en effet concernant le choix et le dimensionnement du réseau, les paramètres à ajuster, le contrôle du système, etc., même avec les outils logiciels de développement maintenant disponibles.
Le chapitre qui suit sera consacré à l'étude des microprocesseurs DSP.

III.1. Introduction

L'utilisation des techniques numériques a nettement pris le pas sur les techniques analogiques vu leurs nombreux avantages qui ont révolutionné et bouleversé l'électronique.

Et ceci dans de nombreux domaines parmi lesquels on cite : le traitement du signal, les télécommunications, le traitement de la parole, les radars, les applications médicales, la commande,...etc.

L'apparition des microprocesseurs standard, suivis des microcontrôleurs, puis des **DSPs** (processeurs pour traitement de signal), a rapidement assuré l'avantage des solutions programmées sur les solutions câblées.

Les DSPs ont pu satisfaire deux objectifs : le calcul **numérique** et la notion du **temps réel.**

Dans ce chapitre il sera question d'une présentation générale des DSPs, puis d'un développement architectural et d'une étude détaillée de leurs différents éléments .Notre étude se concentrera ensuite sur l'étude matérielle et logicielle des DSPs de la firme Texas Instruments « **TMS320C5000** », qu'on a ciblé pour implémenter des routines de traitement d'image. Enfin nous verrons les différents outils de développement mis au service de l'utilisateur.

III.2. Généralités sur les DSPs (Digital Signal Processors)

III.2.1. Qu'est ce qu'un DSP ?

Un DSP (Digital Signal Processor) est un microprocesseur spécialisé. Ce microprocesseur est optimisé pour le traitement numérique du signal en temps réel, c'est-à-dire contenant les fonctions les plus utilisées implémentées (rabaissées) au niveau de la couche matérielle. On le trouve dans de nombreux systèmes embarqués confrontés à un traitement du signal.

III.2.2. Domaines d'utilisation des DSPs

À l'origine, les deux principaux domaines d'applications des DSPs ont été les télécommunications et le secteur militaire. On peut citer : les modems, les codeurs de

parole, les annuleurs d'échos, les récepteurs de numérotation téléphoniques et plus récemment le radiotéléphone.

Aujourd'hui, les applications se sont diversifiées et s'orientent vers le multimédia (cartes sons et cartes vidéo pour PC), l'électronique grand public (CD-ROM, karaoké, télévision numérique, synthétiseurs musicaux), l'informatique graphique, mais aussi l'automatique (surveillance et commande des machines, contrôle des moteurs, robotique), le médical (traitement et archivage d'images, analyse de signaux électrocardiographies, implants cochléaires), l'instrumentation (analyse de spectre, analyse de transitoires, générations de fonctions, interprétation des signaux sismiques), et l'électronique automobile (détection de cliquetis, système ABS). [24]

III.2.3.Particularité des DSPs par rapport aux microprocesseurs?

Un DSP est un type particulier de microprocesseur, il se caractérise par le fait qu'il intègre un ensemble de fonctions spéciales, ces fonctions sont destinées à le rendre particulièrement performant dans le domaine du traitement numérique du signal. Comme un microprocesseur classique, un DSP est mis en œuvre en lui associant de la mémoire (RAM, ROM) et des périphériques. Un DSP typique à plutôt vocation à servir dans des systèmes de traitement autonomes. Il se présente donc généralement sous la forme d'un microcontrôleur intégrant selon les marques et les gammes des constructeurs, de la mémoire des timers, des ports séries synchrones rapides, des contrôleurs DMA, des ports d'E/S divers.

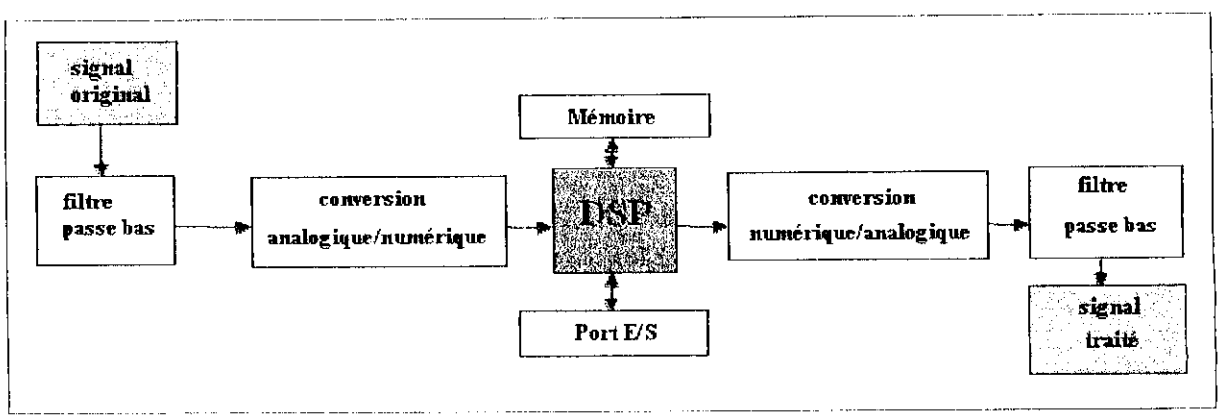


Figure III.1 : Chaîne complète typique d'un système de traitement numérique du signal.

Les particularités de ces puces, par rapport aux processeurs généraux (Intel, Cyrix, AMD...) sont, en général :

➢ **Architecture HARVARD, séparant bus programme et bus données**

L'architecture d'un microprocesseur, et donc d'un DSP, est un élément important qui conditionne directement les performances d'un processeur. Il existe deux types fondamentaux de structures dites « Von Neuman » et « Harvard » :

- **Structure de Von Neuman**

Un microprocesseur basé sur une structure Von Neuman stocke les programmes et les données dans la même zone mémoire .Une instruction contient le code opératoire et l'adresse de l'opérande.

Inconvénient de cette structure : on ne peut lire une donnée ou une instruction en **un seul cycle.**

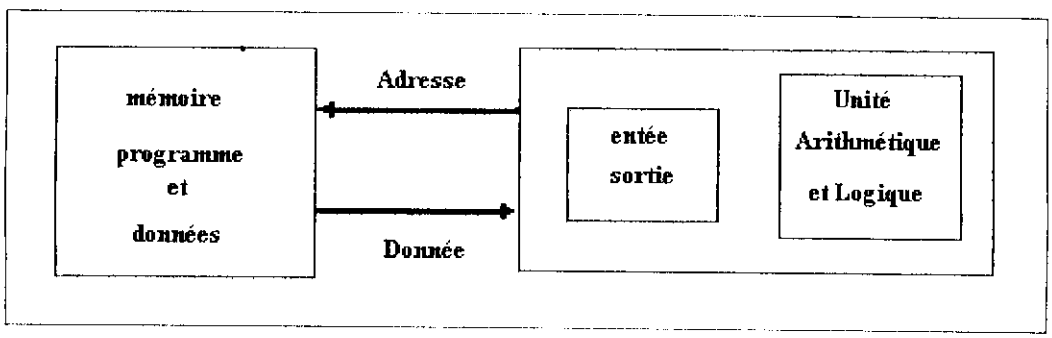


Figure III.2 : Architecture de Von Neuman.

- **Structure d'Harvard**

Cette structure se distingue de l'architecture Von Neuman par le fait que les mémoires programme et données sont séparées, l'accès à chacune des deux mémoires se fait via un chemin distinct. Cette organisation permet de transférer une instruction et des données simultanément, ce qui améliore les performances

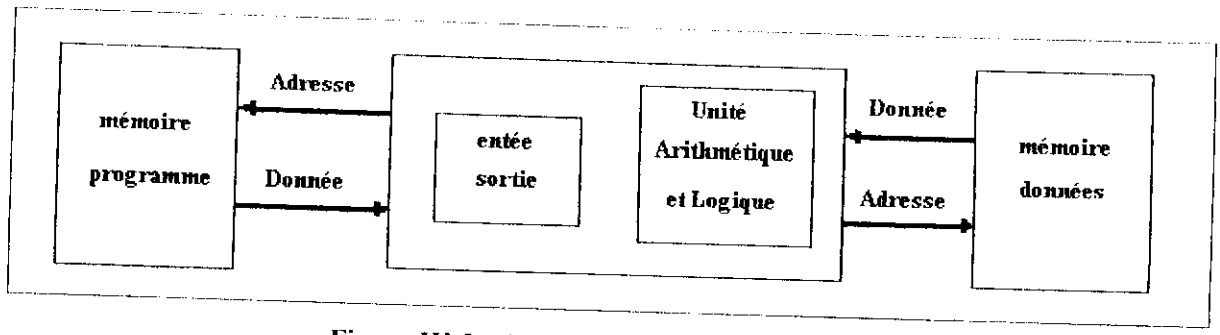


Figure III.3 : Architecture d'Harvard.

Lorsque la vitesse d'exécution d'un programme est importante, il est préférable d'utiliser la structure de Harvard où la mémoire programme et la mémoire données sont séparées. Ainsi, le processeur pourra lire l'instruction et la donnée en un seul cycle. [41] [33]

Généralement, l'architecture utilisée par les microprocesseurs est celle de Von Neuman (exemples : la famille Motorola 68XXX, la famille Intel 80X86).
L'architecture Harvard est plutôt utilisée dans des microprocesseurs spécialisés pour des applications temps réels, comme les DSPs.

- **Architecture de Harvard modifiée**

La structure de HARVARD est cependant pénalisante par le nombre de broches utilisées et le nombre de boîtiers mémoires utilisés. Texas Instrument, comme de nombreux autres constructeurs (Intel avec le MCS 51, Microchip avec les microcontrôleurs PIC) utilise une structure de Harvard modifiée, qui consiste à n'utiliser qu'un bus commun pour les accès en mémoire externe (architecture de Von Neuman) et à utiliser une structure de Harvard pour les accès en mémoire interne. Soit à l'intérieur, la puce DSP dispose de deux bus distincts de données et de deux bus distincts d'adresses. Le transfert des données entre les bus externes et internes est effectué par multiplexage temporel.
Le programmeur est chargé ensuite, si la taille de la mémoire programme ou de la mémoire de données est trop importante, de placer en mémoire interne le bout de programme et de données qui demande une rapidité optimale d'exécution.

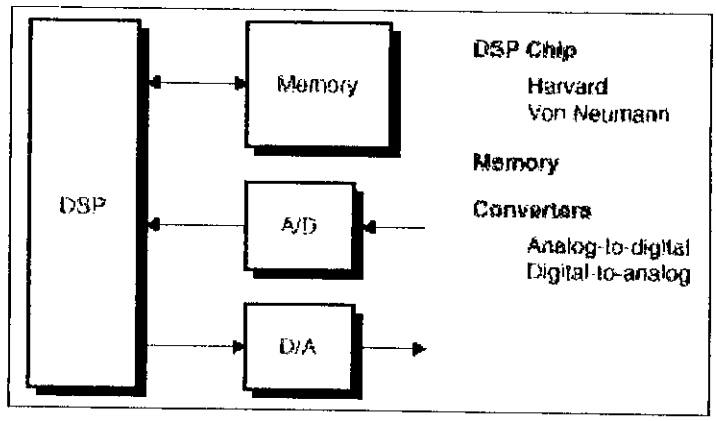


Figure III.4 : Structure de Harvard modifiée (utilisation de la mémoire externe sur un bus par le DSP).

➢ **Instructions adaptées aux opérations classiques de traitement de signaux**

Le DSP a une architecture interne optimisée pour cela il dispose d'une structure de calcul élémentaire où sont implémentées une structure de délai, une multiplication, et une addition : la structure est nommée **MAC (Multiplier et Accumuler).**

L'opération MAC

Après avoir été numérisé, le signal se présente sous la forme d'une suite de valeurs numériques discrètes. Cette suite de valeurs (ou échantillons) est apte à être stockée et traitée par un système .Par nature, le traitement numérique du signal revient essentiellement à des opérations arithmétiques de base du type :

$$A=(B*C)+D$$

Un microprocesseur classique va nécessiter plusieurs cycles d'horloge pour effectuer un tel calcul, par exemple, un 68000 a besoin de :

- 10 cycles d'horloge pour effectuer une addition.
- 70 cycles d'horloge pour effectuer une multiplication.

Soit 80 cycles pour seulement calculer **A**. Si ce temps est admissible dans des applications informatiques courantes, il n'est pas acceptable pour faire du traitement rapide du signal. Les DSPs sont donc conçus pour optimiser ce temps de calcul. À cet effet, ils disposent de fonctions optimisées permettant de calculer **A** beaucoup plus rapidement.

Dans la pratique, la plus part des DSPs ont un jeu d'instructions spécialisé permettant de lire en mémoire une donnée, d'effectuer une multiplication puis une addition, et enfin d'écrire en mémoire le résultat, le tout en un seul cycle d'horloge. [39] [23]

- Les temps d'accès mémoire RAM (Random Access Memory) et ROM (Read Only Memory) sont réduits.

- **Mémoire à accès multiple**

Alors qu'un microprocesseur n'est pas conçu pour une application spécifique, le processeur

DSP est optimisé pour effectuer un traitement numérique du signal. Il dispose notamment d'une mémoire à accès multiple (Figure III.5). Ceci permet de faire des accès simultanés en lecture ou en écriture, grâce à deux bus de données et d'adresse distincts. Le DSP dispose aussi de modes d'adressage particuliers, ainsi que toutes les ressources mémoire intégrées sur la puce.

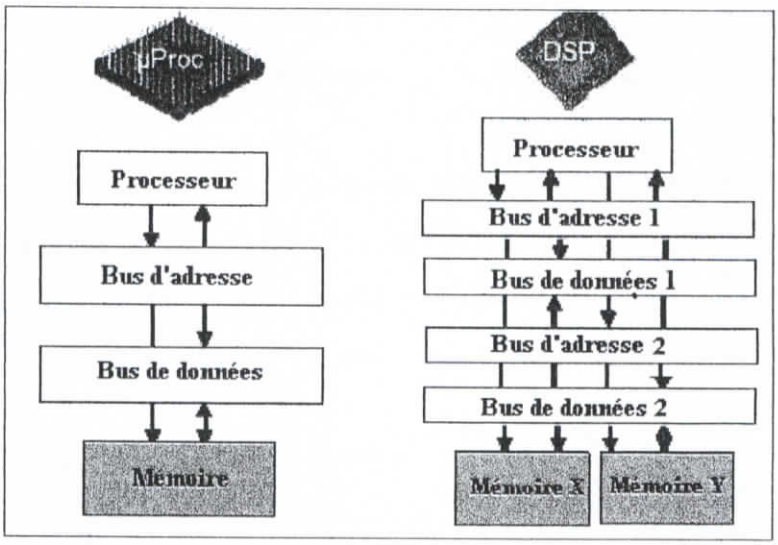


Figure III.5 : Disposition des zones mémoires pour un DSP et un microprocesseur.

- Transferts de données externes par DMA (Direct Access Memory) vers les mémoires sans interférer avec l'exécution du programme.

- Le DSP, contrairement au microprocesseur, intègre des interfaces directes avec les signaux à traiter : interfaces son, vidéo, ou réseau (Figure III.6). Notons aussi que de plus en plus de DSPs sont conçus pour fonctionner en parallèle, traitant seulement une portion du signal au cours du temps, ce qui est équivalent à un multiplexage temporel. Les applications sont principalement les réseaux à très haut débit de données, chaque DSP prenant alors en charge le traitement d'un paquet parmi n.

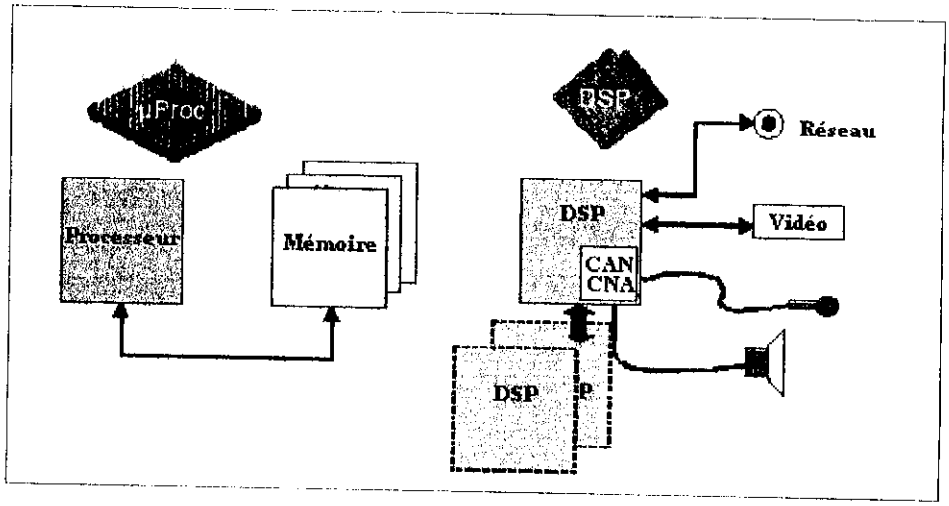


Figure III.6 : Interaction du DSP avec des circuits extérieurs.

- Les DSPs peuvent contenir des convertisseurs analogiques-numériques et numériques- analogiques intégrés. [40]

- **Mode d'adressage particulier**

Les modes d'adressage des données sont un point particulier des DSPs. Un DSP peut posséder plusieurs unités logiques de génération d'adresse travaillant en parallèle avec la logique du cœur DSP. Une unité logique de génération d'adresse est paramétrée une seule fois via les registres appropriés .Elle génère alors toute seule en parallèle avec l'exécution d'une opération arithmétique, les adresses nécessaires à l'accès des données. Nous détaillerons cette partie ultérieurement.

- **Jeu d'instructions et particularités**

Les DSPs sont dotés d'instructions facilitant les traitements les plus couramment rencontrés en traitement numérique du signal : carrés, racines carrées, valeur absolue, etc.

Exemples

- **SQRA** (Square and Accumulate Previous Product).
- **SQRS** (Square and Substract Previous Product) sur le C50.
- **ABS** retourne la valeur absolue.
- **RND** effectue un arrondi.

- Une architecture à pipeline permettant de paralléliser le maximum d'opérations élémentaires (préchargement d'instruction, chargement des données, exécution d'opérations arithmétique/flottante, stockage des résultats). [35]

III.2.4. Principaux constructeurs de DSP

Rang mondial	vendeur	Familles de DSP	Format	Données (bits)	Puissance (MIPS)	Site web
1	Texas Instruments	TMS320C1x	Fixed	16	8.8	dspvillage.ti.com
		TMS320C2x	Fixed	16	40	
		TMS320C3x	Floating	32	25	
		TMS320C4x	Floating	32	30	
		TMS320C5x	Fixed	16	50	
		TMS320C6x	Fixed	32	2400	
2	Analog Devices	ADSP-21xx	Fixed	16	40	www.analog.com
		ADSP-21	Floating	32	40	
3	motorala	DSP560xx	Fixed	24	40	www.motorola.com
		DSP563xx	Fixed	24	80	
		DSP568x x	Fixed	16	70	
		DSP960xx	floating	32	40	
		StarCore	Fixed	16	1200	
4	Lucent	DSP16xx	Fixed	16	40	www.lucent.com
		DSP32xx	floating	32	40	

Tableau III.1 : Principaux constructeurs de DSP [35]

III.3. Les TMS320C5000

Pour l'ensemble de nos implémentations, nous avons utilisé la génération TMS320C5000 de la firme Texas Instruments. C'est les DSPs dont les outils de développement sont disponibles au sein de notre école.

Avant d'aborder l'étude détaillée de ce dernier, on va présenter un aperçu général de la grande famille des TMS320.

III.3.1. généralités sur la famille TMS320

En 1982 Texas Instruments introduit le TMS32010, le premier DSP à virgule fixe de la famille TMS320 .Avant la fin de l'année, le magazine « Electronic Products » a consacré au TMS32010 le titre « de produit de l'année ».

Cette famille est constituée de processeurs à virgule fixe 16 bits (processeurs C1x, C2x et C5x, C54x) et de processeurs à virgules flottantes 32 bits (C3x et C4x) et 64 bits (C8x). Ces processeurs concurrencent directement les applications spécifiques à base d'ASIC. [31]

Les trois grandes classes de cette famille sont :

- Les DSPs à **virgules fixes** : moins chers mais font des erreurs de calcul constantes, ce sont les plus utilisés.
- Les DSPs à **virgules flottantes** : font intervenir une mantisse et un exposant. Dans ce cas, l'erreur par *overflow* est limitée puisque on bénéficie d'une dynamique plus importante. [25]
- Le DSP TMS320C80, processeur à virgule flottante, est utilisé en parallèle avec d'autres C80, pour les applications vidéo.

Maintenant il y a une nouvelle génération de DSP : les **TMS320C6x,** avec une grande performance et des caractéristiques particulières.

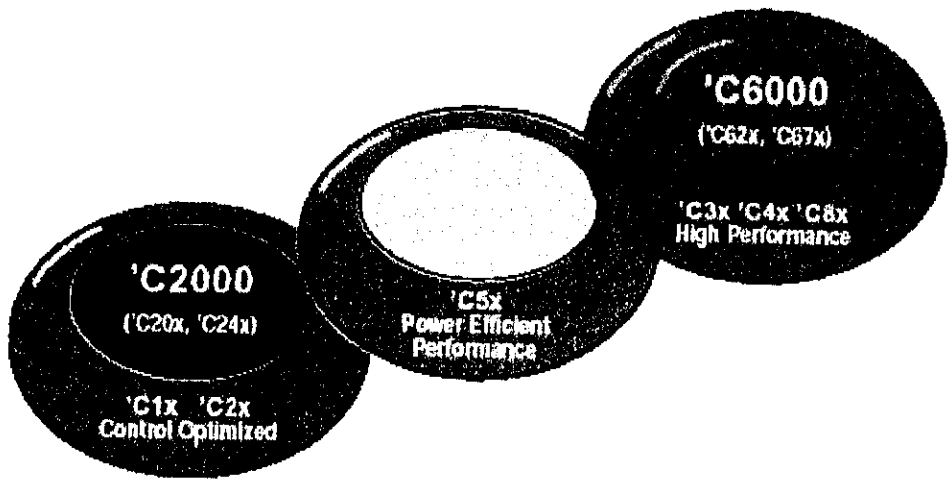


Figure III.7 : La famille des DSP TMS320.

Les avantages de cette famille sont :

-Une faible consommation.

-Les leaders du marché pour les signaux analogues et mixtes.

-Une variété d'options de boîtiers.

-Les bas prix des kits de développement et les modules d'évaluation.

-Des cycles précis de simulation.

-Optimisation du langage de haut niveau du compilateur.

-Utilisation des émulateurs en temps réel.

-Application ou ajout des librairies.

III.3.2. Description générale des C5000

Les initiales de la famille TMS320C5000 signifient :

TMS	320	C	5xx
Préfixe indiquant qu'il s'agit d'un circuit qualifié (recommandé à l'utilisation d'après l'expertise de la firme TI)	La famille	La technologie CMOS	La génération

La familleTMS320C5000 a été créée pour répondre au marché du téléphone mobile qui demandait un DSP à virgule fixe et une faible consommation, avec un jeu d'instructions adapté aux algorithmes du GSM. Plus généralement, ces DSPs sont bien adaptés aux systèmes embarqués temps réel.

Pour les C54x,les temps de cycles vont de 10 à 25 ns (40-100Mhz), la RAM interne de 5 à 32 Kmots, et les périphériques consistent en un ou plusieurs ports série, un port d'interface vers un processeur hôte et un timer. La ROM est également de taille variable. Le processeur est physiquement disponible en boîtier TQFP de 100,128 ou 124 broches selon le modèle. Ces DSPs ont des zones mémoire de données et programme séparées.

Le C55x représente la dernière génération des DSPs C5000 virgule fixe. Son code source est compatible avec celui des C54x .Le cœur de ce DSPs délivre 5 fois plus de performances et dissipe 1/6 moins d'énergie que le C54x. Il a une zone mémoire données et programme unifiée. Il peut atteindre la fréquence de 300 Mhz.

Tous les circuits de la chaque famille C54x, respectivement C55x, ont le même CPU, ils différent par la taille de la mémoire interne et par le type des périphériques intégrés. [24] [26]

	'C54x	'C55x
MACs	1	4
Accumulators	2	3
Read buses	2	2
Write buses	1	1
Program fetch	1	6
Address buses	4	8/16/24/32/40/48 bits
Program word size	16 bits	16bits
Data word size	16 bits	3(24-bit each)
Auxillary registers ALUs	2(16-bit each)	1(40-bit)
ALU	1(40-bit)	1(16-bit)
Auxillary registers	8	8
Data registers	0	4
Memory space	Separate program/data	Unified space

Tableau III.2 : Comparaison entre les deux générations des C5000.

III.3.3. Etude matérielle (hardware)

III.3.3.1. Architecture

Ces DSPs ont des bus de données et d'adresses de 16 bits chacun. Ils utilisent une architecture Harvard modifiée qui maximise la capacité de traitement.
Ces processeurs ont une unité arithmétique et logique (UAL) à haut degré de parallélisme, une logique hardware d'application spécifique, mémoire interne et des périphériques externes. Ils ont aussi un ensemble d'instructions spécialisé, qui est la base de la flexibilité opérationnelle et la rapidité de ces DSPs. Séparer les espaces donnée et programme permet un accès simultané aux instructions du programme et aux données et donc par conséquent un haut degré de parallélisme. Deux opérations de lecture et une opération d'écriture peuvent être effectuées en un seul cycle. Les instructions avec emmagasinement parallèle et les instructions spécifiées peuvent pleinement utiliser cette architecture. Ajouté à cela le fait que les données peuvent être

transférées entre espaces données et programme. Ce parallélisme appuie un ensemble d'opérations de manipulation de bits qui peuvent être tous manipulés dans un seul cycle machine. Aussi sont inclus les mécanismes de contrôle pour diriger les interruptions, les opérations répétées et les fonctions d'appel. [27]

III.3.3.2. L'unité centrale de calcul (CPU)

L'unité centrale de calcul (CPU) comprend :

- Une unité logique et arithmétique de 40 bits pour le C54x et deux UALs sur 40/16 bits respectivement pour les C55x.
- Deux accumulateurs 40 bits pour le C54x et 4 accumulateurs pour le C55x.
- Un additionneur/multiplieur 17×17 bits.
- Une unité de comparaison, sélection et stockage (CSSU).
- Un registre à décalage.
- Une unité de génération d'adresses de données et de programme. [27] [24]

III.3.3.3. L'unité arithmétique et logique ALU

Les C54x a une UAL qui effectue des calculs arithmétiques et logiques sur 40 bits et comporte deux accumulateurs (ACCA et ACCB).Le C55x comporte deux UAL l'une sur 40 bits et l'autre sur 16 bits et quatre accumulateurs.

L'UAL peut fonctionner dans un mode particulier sur 2 mots de 16 bits et effectuer alors des opérations duales (2 additions par exemple) simultanément. Ce mode est activé en positionnent le bit C16 de ST1 du registre d'état. [24] [27]

III.3.3.3.1. Les accumulateurs A et B

Les accumulateurs servent de destination pour l'UAL aussi bien que pour le multiplieur/additionneur. [24] [27]

III.3.3.3.2. Le multiplieur additionneur

Le multiplieur du DSP C54x possède un additionneur dédié, cela lui permet l'exécution d'une instruction MAC en un seul microcycle .Tandis que le C55x contient deux unités MACs effectuant la multiplication et l'adition/soustraction en un seul cycle, chaque MAC peut effectuer une multiplication 17×17 bits (réel ou entier) et une adition 40 bits. [24]

III.3.3.4. Les registres

- Registre à décalage : situé au cœur de la CPU, sert à cadrer les données venant de la mémoire ou bien la valeur de l'accumulateur avant une opération dans l'UAL. Il permet au processeur d'effectuer un encadrement numérique de l'accumulateur, extraction de bits, arithmétique poussée et des opérations de prévention d'overflow.
- Les registres d'état et de contrôle : Le C54x possède trois registres de contrôle : ST0, ST1 et PMST. Les deux premiers servent à configurer le processeur et PMST est utilisé pour la configuration et le contrôle de la mémoire. Le C55 x possède quatre registres d'état.
- Registre compteur programme sur 24 bits.
- Les registres auxiliaires (AR0-AR7).
- Registres temporaires (TREG).
- Registres de transition (TRN).
- Registre du pointeur de pile (PS).
- Registre circulaire (BK).
- Registres block répétition (BRC, RSA, REA).
- Registres d'interruption (IMR, IFR).

Selon la version du circuit, le type et la taille de la mémoire intégrée dans le circuit diffèrent. [24] [27]

III.3.3.5. Architecture des bus

Le processeur C54x comporte quatre principaux accès aux données : [24]

- Un bus programme PB qui véhicule les instructions et les coefficients stockés dans la mémoire programme.
- Trois bus d'accès aux données sont raccordés à l'unité centrale de calcul, à la mémoire de données, aux périphériques et aux circuits de génération d'adresse (programme et données)

Le processeur C55x comporte : [28]

- Un bus programme
- Trois bus de lecture de données
- Deux bus d'écriture de données
- Un bus additionnel pour les périphériques et les transitions DMA.

III.3.3.6. Mémoire

Le DSP peut adresser un espace mémoire de 192 Kmots de 16 bits (8 Mmots) incluant mémoire interne et externe .Cet espace correspond à trois espaces spécifiques séparés de 64 Kmots de mémoire programme, 64 Kmots de mémoire de données (data) et 64 Kmots d'entrées/sorties pour le C54x et espace unifié programme/donnée pour le C55x. Chaque circuit comporte de la RAM à double accès (DRAM), de la RAM à simple accès (SRAM) et de la ROM interne. La RAM est généralement réservée aux données mais elle peut aussi être configurée pour faire partie de la zone programme. [24] [28]

- **La DARAM interne :** RAM à double accès prend en charge deux accès (lecture et écriture) par bloc mémoire en un cycle unique.

- **La SARAM interne :** RAM à accès unique prend en charge (lecture ou écriture) par bloc mémoire en un seul cycle.

- **ROM interne :** La ROM offre un stockage de mémoire non volatile pour le programme et la donnée. Cette ROM est à accès unique, seulement une lecture peut être effectuée en un temps. Chaque accès ROM requiert deux cycles.

Selon la version du circuit, le type et la taille de la mémoire intégrée dans le circuit diffèrent.

III.3.3.7. Le pipeline

Les opérateurs de la CPU sont conçus pour effectuer une opération par temps de cycle.

Les DSP C5000 exécutent une instruction à travers les étages du pipeline suivants :

- **Etage FETCH :** lit les données programme de la mémoire vers la queue du buffer d'instruction.
- **Etage DECODE :** décode les instructions et dispatche les taches aux autres unités fonctionnelles primaires.
- **Etage ADRESS :** calcule les adresses pour les accès données et branche les adresses pour les discontinuités du programme.
- **Deux étages ACCESS1/ACCESS2** : envoient les adresses de lecture des données à la mémoire.

- **Etage READ :** transfert l'opérande sur le bus B, C et D.
- **Etage EXECUTE :** exécute l'opération dans l'unité A et D et écrit dans les bus E et F. [28]

III.3.3.8. Les périphériques

Les processeurs C5000 disposent de plusieurs périphériques intégrés auxquels sont associés des registres de contrôle configurés en mémoire. [27]
Les périphériques disponibles dépendent de la version du circuit et comprennent :

- Un générateur de temps d'attente.
- Un banc de changement (switching) programmable.
- Ports d'entrée sortie parallèle.
- Contrôleur DMA.
- Interface port- hôte (standard 8 bits, amélioré 8 bits et 16 bits).
- Ports série.
- Broches d'E/S universels.
- Un timer 16 bits.
- Générateur d'horloge de phase en boucle fermée (PLL).

III.3.4. Etude logicielle (software)

III.3.4.1. Les modes d'adressage

La puissance d'un DSP est liée en partie à ses modes d'adressage, qui permettent un codage des adresses et de l'instruction sur un seul mot (16 bits), et rendent ainsi l'exécution possible en un temps de cycle. Cependant, certains modes d'adressage se font obligatoirement sur 2 mots, comme par exemple les initialisations de registres. Les C5xx comportent 7 différents types d'adressage : [24] [23]

- **Adressage immédiat**, où la valeur de l'opérande est donnée dans l'instruction.

  Une valeur est spécifiée sur 3, 4, 8 ou 9 bits pour un adressage court (instruction

  codée sur un mot) ou 16 bits pour un adressage long (instruction codée sur 2 mots) .Ce type d'adressage est utilisé lorsque l'on souhaite initialiser un registre.

- **Adressage absolu**, où l'adresse de l'opérande est donnée dans l'instruction.

- **Adressage par l'accumulateur,** le contenu des 16 bits de poids faibles de l'accumulateur sert d'adresse (adresse calculée) qui pointe vers la mémoire programme.
- **Adressage direct**, où une partie de l'adresse de l'opérande est donnée dans l'instruction, le reste étant dans un pointeur de page par exemple, c'est un adressage par page de 128 mots (7 bits d'adresse pour une page), la page est sélectionnée par un pointeur sur 9 bits (512 pages).
- **Adressage indirect**, où les adresses des opérandes sont stockées dans des registres auxiliaires. Il inclut :
  1- **L'adressage circulaire**, les calculs sur les adresses contenues dans les registres auxiliaires se font modulo une certaine valeur (utile pour l'implémentation de ligne à retard).
  2- **L'adressage bit-reverse**, utilisé pour les calculs d'adresse de la FFT.
  3- **L'adressage indirect à deux opérandes,** les instructions à deux opérandes permettent d'effectuer deux lectures et une écriture en un seul cycle.
- **Adressage des registres situés dans la page 0,** il est possible d'accéder directement aux registres recopiés dans la mémoire de données de la page 0. Cela se fait par une lecture ou une écriture mémoire.

- **Adressage de la pile,** lors de l'appel à un sous programme ou lors de la validation d'interruption, le processeur sauvegarde la valeur du PC (compteur programme) dans la pile, adressée par le pointeur de pile SP.

III.3.4.2. Les interruptions

Ces DSPs possèdent quatre broches externes d'interruption $\overline{INT0}$ à $\overline{INT3}$ qui sont mémorisées sur le DSP, ce qui permet un fonctionnement asynchrone. A ces interruptions d'usage général, il faut ajouter une broche de reset $\overline{RS}$ et une broche d'interruption non masquable $\overline{NMI}$ (Non Masquable Interupt).
Ces interruptions externes peuvent également être forcées par programme grâce à l'instruction *INTR*. Par ailleurs, des interruptions internes peuvent être générées par des périphériques intégrés : le timer génère $\overline{TINT}$, les ports série génèrent $\overline{RINT0}$, $\overline{XINT0}$, $\overline{RINT1}$, et $\overline{XINT1}$.

Les broches interruptions externes sont échantillonnées plusieurs fois pour éviter les interruptions générées par du bruit. Il faut que le signal soit à l'état bas pendant trois cycles consécutifs pour qu'il soit pris en compte. Une fois qu'une interruption externe est détectée, les C5xx doivent échantillonner un signal à l'état haut au moins pendant deux cycles consécutifs avant de pouvoir détecter une nouvelle interruption. Les broches d'interruptions sont échantillonnées sur le front montant de CLKOUT1. Si ces interruptions fonctionnent de façon asynchrone, il faut allonger les impulsions sur au moins trois cycles pour garantir trois cycles à l'état bas.
Les interruptions sont vectorisées, la plus prioritaire est le Reset. [24]

III.3.4.3.Types d'opérandes et d'opérateurs

Les unités de calcul des DSP travaillent soit en format fixe, soit en format flottant pour les données réelles. Les opérateurs format fixe sont plus rapides et consomment moins d'énergie que les opérateurs format flottant, mais la programmation en format fixe est plus délicate.
La plateforme des C5000 est à format virgule fixe.

III.3.4.4. Types de données

On rappelle que les types *char*, *short*, *int* sont équivalents et correspondent à des valeurs sur 16 bits. Il en est de même pour les types unsigned short. On remarque donc que le type char est signé et sur 16 bits de façon à ce qu'on puisse l'adresser individuellement. [24]
Les types *long* et *unsigned long* correspondent à des valeurs sur 32 bits. Les types signés sont en complément à 2.
Les types pointeurs sont sur (16/23/24 bits).
Un nombre réel X est représenté sur 32 bits (k bits derrière la virgule) ainsi :

$$X \rightarrow \underbrace{b_{32-k-1}\ldots b_1 b_0}_{\text{Partie entière}} \ , \ \underbrace{b_{-1}b_{-2}\ldots b_{-k}}_{\text{Partie fractionnaire}}$$

III.3.5. Outils de génération du code et de développement

L'utilisation des DSP étant relativement sophistiquée, il est important de disposer d'outils de développement performants et conviviaux qui permettent de minimiser le temps de mise sur le marché d'une nouvelle application. Texas Instruments a développé une série d'outils logiciels et matériels, ainsi qu'un système d'aide et d'information à l'utilisateur, exploitant un site très riche. Cet ensemble est enrichi par de nombreux produits proposés par des tierce-parties.

Certains de ce outils offrent notamment un environnement de développement intégré matériel et logiciel : IDE (Integrated Development Environnement). [24]

III.3.5.1. Outils de développement logiciel

Les principaux outils de développement logiciel de cette famille sont listés dans le tableau qui suit. Il fonctionnent généralement sur PC Windows ou NT, et parfois sur station HP ou SPARC. Ces outils permettent de développer du code, de le simuler et de le déboguer. [24]

Outils de développement logiciel	Assembleur, éditeur de lien
	Compilateur C
Outils de simulation et de déboggage	Simulateur et interface de déboggage DOS ou Windows
	Environnement de développement intégré : Code Composer

Tableau III.3 : Outils de développement logiciels.

III.3.5.1.1. Fichier source assembleur/C

Un fichier source assembleur est créé à l'aide d'un éditeur de texte. Il comprend des lignes d'instructions, de directives assembleur, de directives de macro et de commentaires. L'extension de ce fichier est « .asm ». Un fichier source écrit dans le langage évolué C porte l'extension « .c » et est traduit en assembleur par le compilateur C.

La programmation C est intéressante pour réduire le temps de développement d'une application, sa portabilité dans d'autres systèmes et elle ne nécessite pas une connaissance approfondie de l'architecture interne du DSP. On l'utilise pour les parties

de programme dont le temps d'exécution n'est pas critique mais elle occupe plus d'espace mémoire et donc est moins optimisée. On utilise la programmation en assembleur pour les blocs de programmes effectuant des calculs arithmétiques intensifs. [24] [31]

III.3.5.1.2. Assembleur et éditeur de liens

Après avoir écrit un programme à l'aide d'un éditeur de texte (en assembleur ou en C), il faut assembler puis lier le programme pour obtenir un programme exécutable. [24] [31]

a) **L'assembleur :** Traduit les fichiers sources écrits en langage assembleur du DSP en un fichier objet en langage machine au format COFF (Common Objet File Format). Les fichiers source peuvent contenir des instructions, des directives assembleur, des macro directives (soubroutines ou fonctions). Les directives assembleur peuvent être utilisées pour contrôler différents aspects du processus d'assemblage : le format listing du code source, l'alignement des données, le contenu des sections.

b) **L'éditeur de liens (Linker) :** L'éditeur de liens combine plusieurs fichiers objet relogeables en format COFF(créés par l'assembleur) en entrée, pour générer un fichier COFF exécutable par un DSP. Il effectue les relocations et résout les problèmes de références externes. Les directives de cet éditeur nous permettent de combiner les sections du fichier objet, lier les sections du fichier objet, lier les sections ou les symboles aux adresses ou dans les rangs de la mémoire et définie et redéfinie les symboles globaux.

c) **Le fichier en format COFF et sections associées :** Les outils de langage assembleur créent et utilisent les fichiers objets dans le format COFF (Common Object File Format) pour faciliter la programmation modulaire. On peut programmer efficacement les DSP si l'on comprend les bases du format COFF. Les fichiers objets contiennent des blocs séparés (appelés sections) de code et de données que l'on peut charger dans l'espace mémoire du DSP. Une section est un bloc de code ou de données qui va occuper un espace dans la mémoire.

Un fichier COFF contient trois sections par défaut :

1- **Sections initialisées**, comprennent:

« .text » : contient le code exécutable

« .data » : contient les données

2- **Sections non initialisées** :

« .bss » : reserve l'espace pour les variables non initialisées.

```
/*************************************************************/
/*          fichier de commande (linker command file)        */
/*************************************************************/
 Memory
{
   VECS   : origin=00000000h   lenght= 40h
   ROM    : origin=00000040h   lenght= FC0h
   RAM0   : origin=00801000h   lenght= 400h
   RAM1   : origin=00801400h   lenght= 400h
 }


{
  vectors   :  load=0000000h
  .text     :  load = ROM
  .data     :  load =ROM
  .bss      :  load =RAM0
  newvars   :  load =RAM1
 }
```

Figure III.8 : Exemple d'un fichier de commande (définissant les sections et leurs plages en mémoire).

Donc l'éditeur de lien recombine les sections de mêmes noms venant des fichiers objet que l'on veut lier, puis attribue une adresse en mémoire pour cette section. Deux directives lui permettent d'effectuer cette tache :

- **La directive MEMORY :** décrit la configuration de la mémoire physique disponible dans le système cible. Elle permet de donner des noms aux différentes parties de cette mémoire.
- **La directive SECTIONS :** spécifie comment l'éditeur de liens doit combiner les sections de même nom venant de fichiers objets différents et à quelle adresse ou à quelle zone mémoire les affecter.

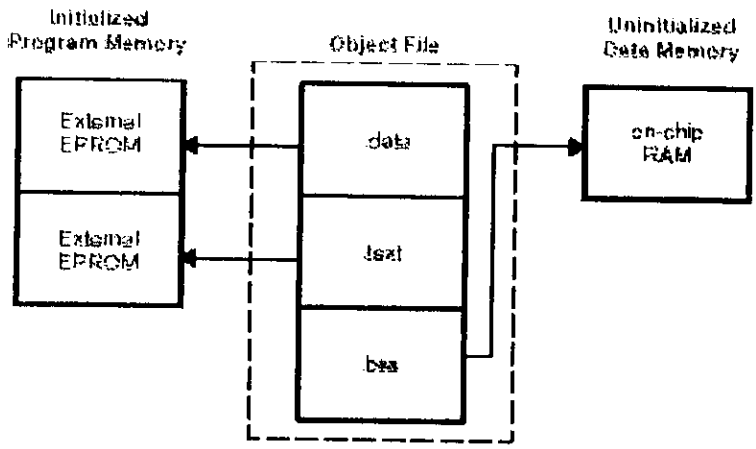


Figure III.9 : Sections existantes par défaut et l'effet du fichier de commande.

Les DSPs de cette famille ont des blocs mémoire séparés, programme, données et entrées/sorties, de même plage d'adresse.

III.3.5.1.3. Utilitaire de construction de librairies (archiver)

Le compilateur C comporte des librairies utiles pour le programme source .On inclue une librairie par la directive « #include » déclarée dans l'entête du fichier.
L'archiver nous permet de rassembler un groupe de dossiers dans un dossier d'archive seul.
Par exemple, on peut rassembler ensemble plusieurs macros dans une bibliothèque du macro. On peut aussi l'utiliser pour rassembler un groupe de fichier objet en une librairie d'objet. L'éditeur de liens va inclure les membres de la librairie qui sont confrontés à des référencements externes durant l'édition de lien. [24] [31]

III.3.5.1.4. Les outils de déboggage software

a) Le simulateur

C'est un programme software qui permet de simuler l'exécution d'un programme écrit en C ou en assembleur pour un DSP. Le simulateur peut exécuter les fichiers objet COFF linkés. Il permet la vérification d'un programme sans les aspects temps réel. La simulation logicielle est effectuée sur une machine d'usage général, un PC ou une station de travail et les entrées/sorties utilisent en général des fichiers.

b) L'environnement de développement intégré Code Composer

C'est un environnement de développement intégré(IDE :Integrated Development Environment) , commun aux différents DSPs de Texas Instruments, qui peut s'utiliser seul ou en lien avec une cible matérielle fonctionnant en temps réel telle qu'une carte d'évaluation . Cet environnement permet de construire un projet, d'éditer les fichiers, de les compiler ou les assembler, de faire l'édition de liens, de tester et de débogger une application mono ou multiprocesseur. Les cibles matérielles supportées sont les cartes d'évaluation EVM, les kits de développement DSK , ainsi que les émulateurs. [23] [24] [31] [35]

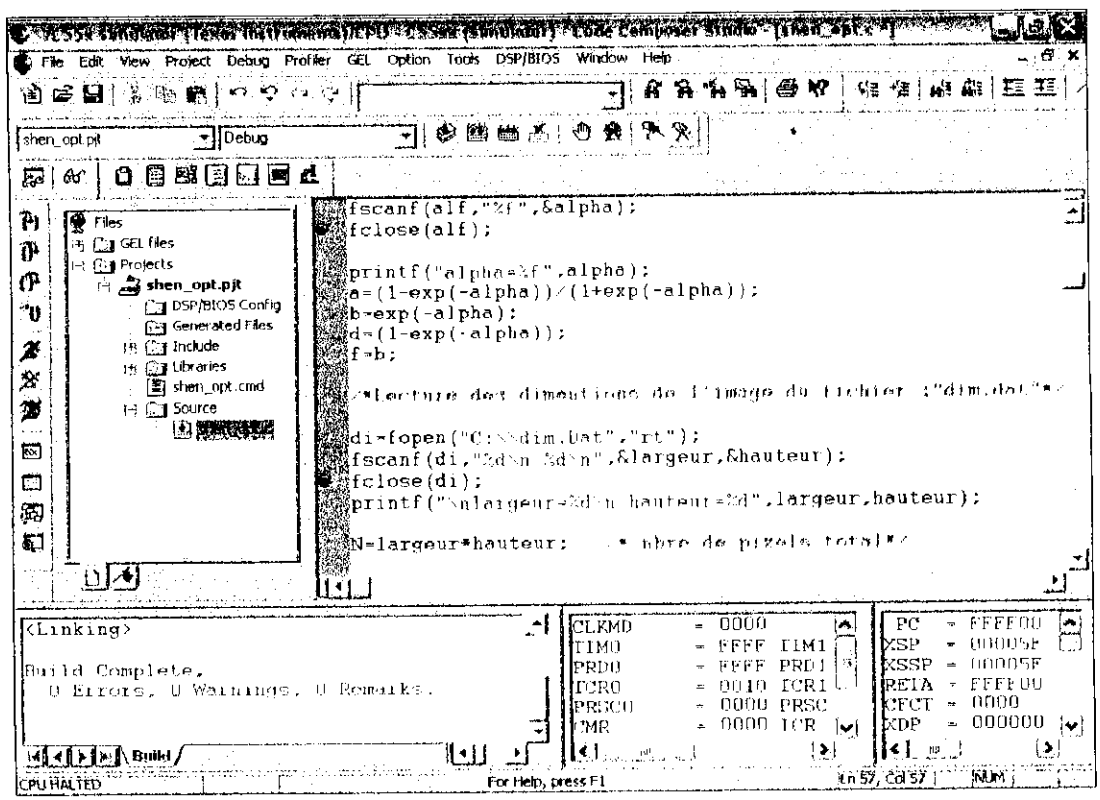


Figure III.10 : Interface du Code Composer

III.3.5.1.5. L'utilitaire de conversion hexadécimale

Le DSP accepte en entrée les fichiers COFF mais ce n'est pas le cas de la majorité des programmeurs d'EPROM. Cet utilitaire convertit le format du fichier COFF en un format pour d'autres circuits .Le fichier converti peut être chargé en EPROM.

III.3.5.2. Outils de développement matériel (hardware)

À ces outils logiciels, s'ajoutent des systèmes matériels permettant de tester une application en temps réel .On présentera ici la carte d'évaluation EVM et le kit de démarrage DSK ainsi les outils d'émulation XDS510. [24] [29] [31]

III.3.5.2.1. Le kit d'évaluation DSK (DSP Starter Kit)

C'est un système à très faible coût, contenant une carte DSP, une alimentation, un débogger et des outils de génération de code assembleur spécifiques au système DSK. La carte se connecte à un PC par un port parallèle.
Elle contient un DSP fonctionnant à 40 Mhz et une interface analogique /numérique et numérique/analogique et bien adapté aux signaux de parole.

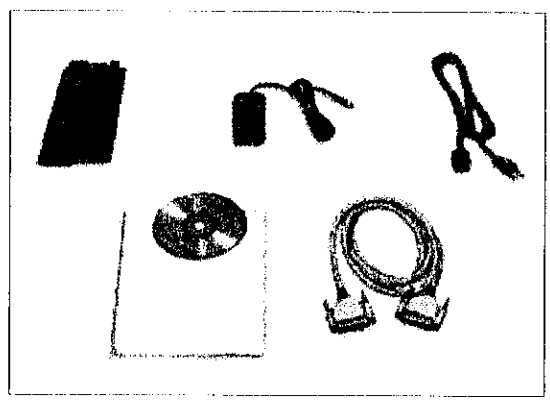


Figure III.11: Kit d'évaluation DSK.

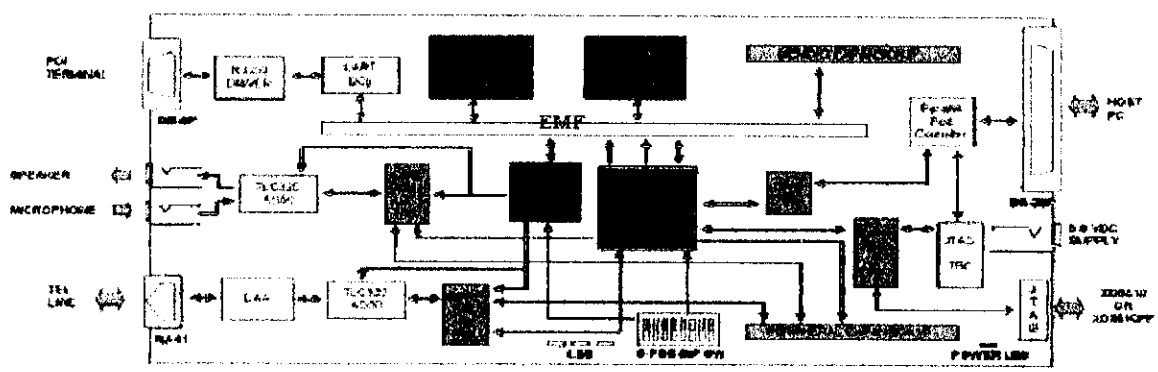


Figure III.12 : Carte DSK.

III.3.5.2.2. La carte d'évaluation

Le module d'évaluation EVM est un système faible coût comprenant une carte PC ISA 16 bits avec un DSP, un débogger et des outils standards de génération de code assembleur fonctionnant sous Windows 95,98 et NT. L'interface graphique du débogger est identique à celle du simulateur. C'est une carte interne au PC qui inclue un DSP, une interface de conversion analogique/numérique et numérique/analogique, une mémoire RAM, une interface avec le PC et une interface d'émulation. Il offre une rapidité dans l'émulation et le déboggage hardware.

III.3.5.2.3. Outils d'émulation

Les outils d'émulation servent à mettre au point une application pour laquelle l'utilisateur a développé le matériel et le logiciel. Il faut dans ce cas tester à la fois le comportement du logiciel et du matériel. Le système XDS510 (Extended Development Support) est un système d'émulation pleine vitesse qui se compose d'une carte JTAG insérable dans un bus PC et d'un câble JTAG. Il permet au débogger de contrôler le DSP sur une carte matérielle, cette carte étant connectée à l'émulateur par le câble d'émulation JTAG relié au port JTAG du DSP.
Cet émulateur permet de démarrer l'exécution d'un programme, de l'arrêter, de faire du pas à pas, de mettre des points d'arrêt logiciels et de faire une trace à n'importe quelle adresse programme ou donnée, de charger et inspecter les registres, les mémoires ; mesurer le temps d'exécution d'un programme.
Il existe sur PC et sur station SPARC. Il fonctionne avec le débougger Code Composer ou le débogger TMS320.

III.4. Conclusion

Les DSPs sont des processeurs dédiés signal qui ont révolutionné les systèmes embarqués. Ceci grâce à leur architecture particulière et leurs périphériques intégrés qui leur procurent puissance et rapidité. Les DSPs TMS320C5000 sont dédiés pour le traitement de la parole et sont à faible consommation .Ce sont des DSPs travaillant en virgule fixe pour le codage des données réelles et ont des bus de données et d'adresses de 16 bits.
Le constructeur Texas Instruments a mis au service de l'utilisateur un ensemble d'outils de développement logiciel et matériel riche et simplifié.

IV.1. Introduction

A l'issue de l'étude détaillée des chapitres précédents, on a pu acquérir les théories du traitement d'images, des réseaux de neurones, ainsi que celles des microprocesseurs DSP.

Le but de notre travail est d'implémenter sur DSP des algorithmes de filtrage optimal appliqués sur des images en niveaux de gris, sans utilisation des réseaux de neurones en premier lieu, puis en introduisant les réseaux de neurones.

Le langage de programmation choisi est le « langage C », vu sa simplicité et sa portabilité dans d'autres systèmes. Afin de nous assurer de l'efficacité de nos filtres, nous avons d'abord testé nos algorithmes en les implémentant en Borland C, ensuite nous les avons transposé sur le DSP.

Le DSP utilisé pour notre implémentation est celui de la famille Texas Instruments «TMS320C5000 ». Ses outils de développement logiciel (simulateur, environnement de développement intégré Code Composer), nous offrent la possibilité de simuler l'exécution d'un programme écrit en C, en C++, ou en assembleur et de le vérifier sans l'aspect « temps-réel » ; ceci pouvant être effectué sur une machine d'usage général, un PC ou une station de travail.

Aussi, ses outils de développement matériel (carte d'évaluation EVM, kit de démarrage DSK, outils d'émulation XDS510), permettent de tester une application en temps-réel. Ceci ne fera pas l'objet de notre travail, seule la simulation sur le Code Composer sera effectuée.

Pour notre cas nous avons utilisé un PC PENTIUM III 994 MHz avec 128 MO de RAM.

Au cours de ce chapitre, nous allons présenter les différents outils et les différentes méthodes utilisées pour notre travail.

Le schéma synoptique général (figure IV.1) permettra de mieux comprendre et situer les différentes parties réalisées dans notre travail :

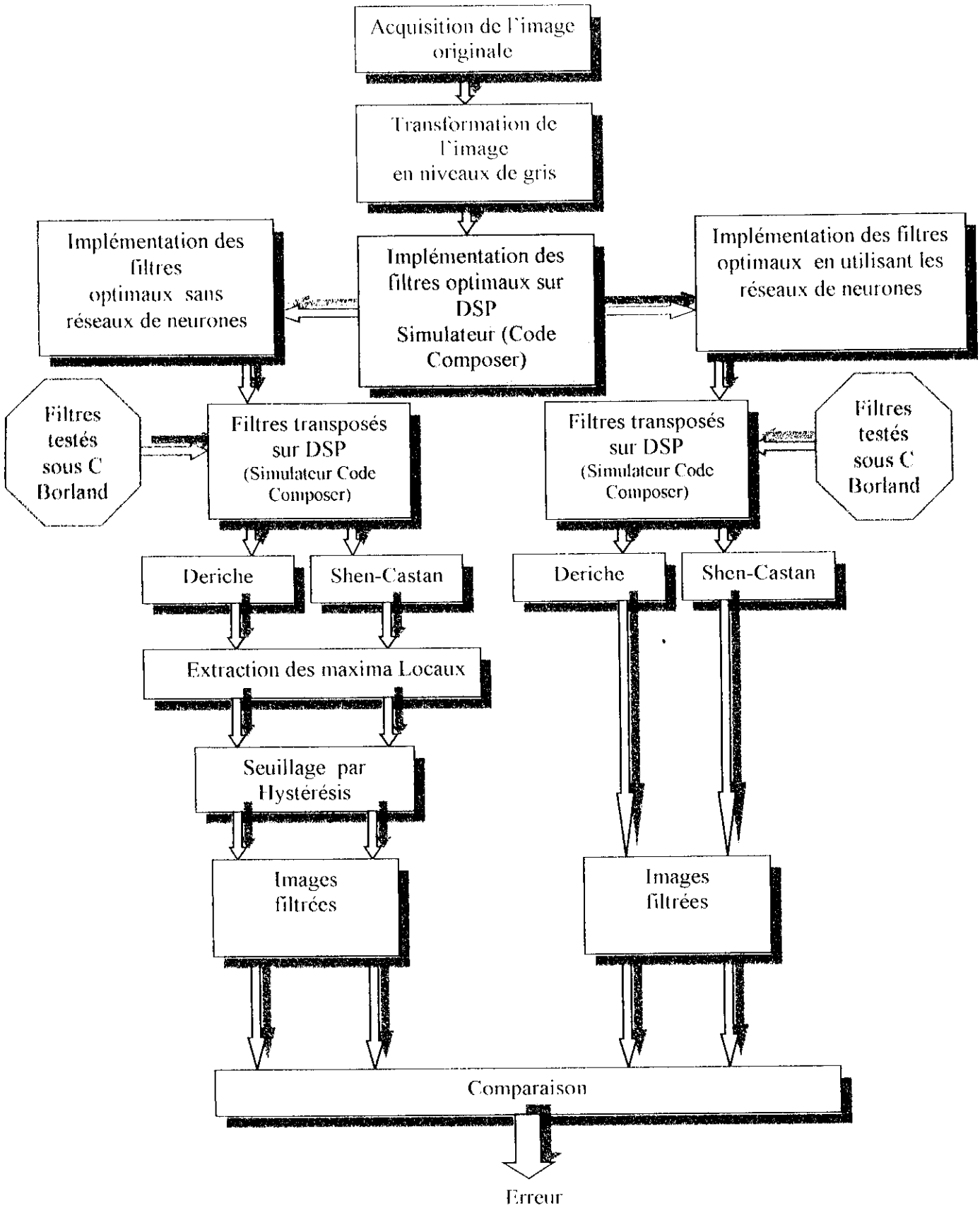


Figure IV.1 : Synoptique du travail effectué.

IV.2. Présentation de chaque étape

IV.2.1. Acquisition de l'image

L'acquisition de l'image constitue un des maillons essentiels de toute chaîne de conception et de production d'images. Pour pouvoir manipuler une image sur un système informatique, il est avant tout nécessaire de lui faire subir une numérisation. Cette étape peut être faite matériellement à l'aide de périphériques spécifiques tels que le scanner et la camera. Ainsi, nos images ont pu être sauvegardées et stockées sous plusieurs formats de fichiers tels que : Bmp, Jpeg, Gif...etc. (Voir annexe 1).

IV.2.2. Niveaux de gris

Après numérisation de nos images, il est nécessaire de les transformer en niveaux de gris afin de pouvoir leurs appliquer le filtrage optimal. Dans ce cas la couleur d'un pixel peut prendre des valeurs allant du noir au blanc, en passant par un nombre fini de niveaux intermédiaires. (Voir chapitre I.4.5). Dans notre cas, on obtient 256 teintes de gris.

La valeur de chaque pixel de l'image sera calculée avec l'équation fondamentale de la télévision :

$$N = 0{,}3 * R + 0{,}59 * V + 0{,}11 * B \quad \ldots\ldots\ldots (37)$$

IV.2.3. Filtrage optimal sur DSP

Le but du filtrage est d'améliorer la qualité de l'image en atténuant le bruit. Parmi les différents filtres étudiés, le « filtre optimal » fait l'objet de notre travail.

Ce filtre a pour but l'extraction de l'attribut « contour » correspondant à des points d'intérêt de l'image. Les filtres implémentés sont les filtres de lissage et de dérivation de Shen-Castan d'une part et de Deriche d'autre part, en se basant sur l'étude de Canny. La détection de contours impliquera dans ce cas la recherche des discontinuités locales de la fonction des niveaux de gris de l'image en utilisant un filtrage récursif.

Le schéma de la figure IV.2 nous montre le principe de fonctionnement du filtre optimal :

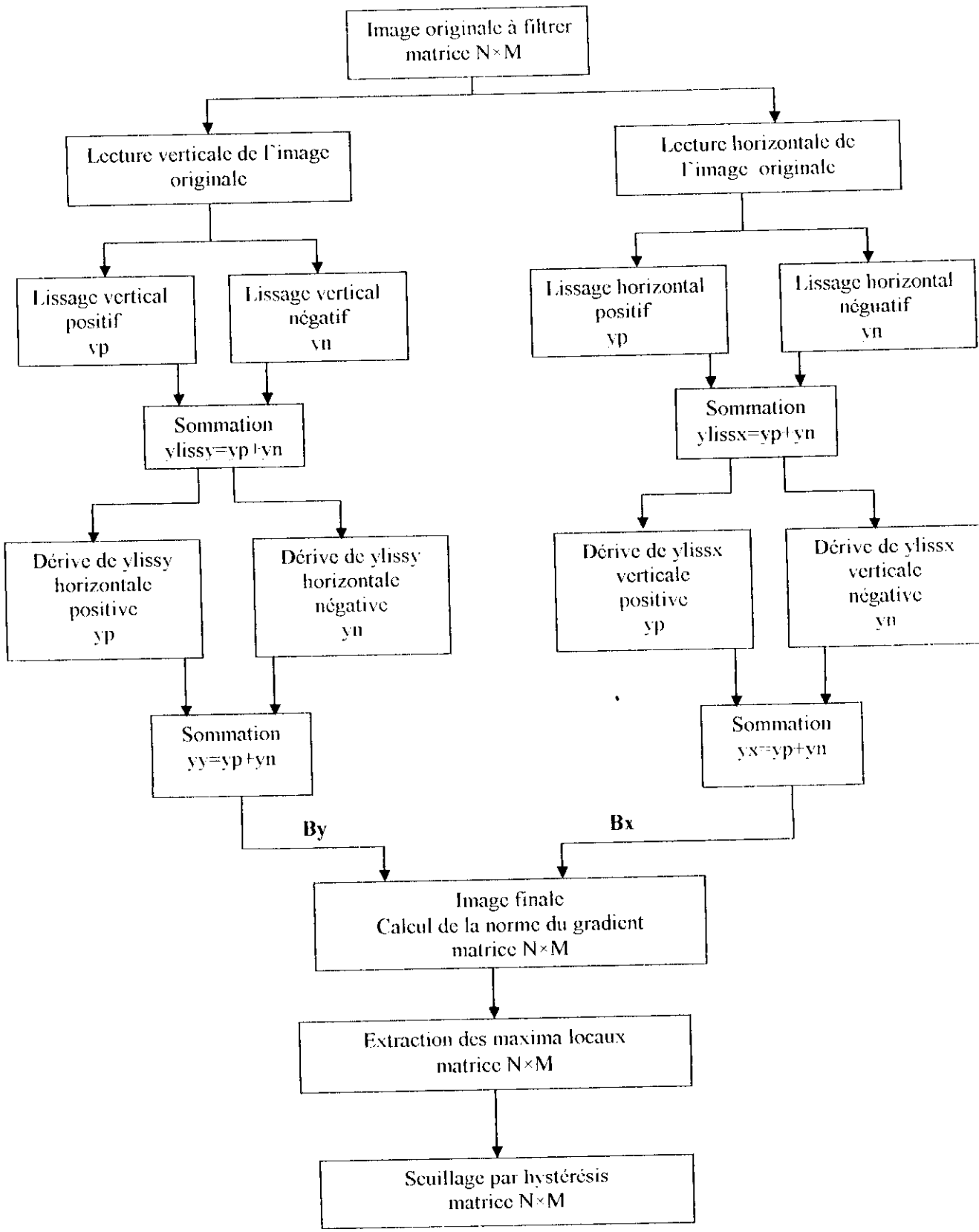


Figure IV.2 : Schéma de fonctionnement du filtre optimal (Deriche et Shen).

Pour l'implémentation de ces deux filtres optimaux (Shen et Deriche) sur DSP, nous avons choisi deux méthodes, testées d'abord sous le Borland C, qui sont :

- Une implémentation sans les réseaux de neurones ;
- Une implémentation en utilisant les réseaux de neurones.

Tests sous Borland C

Simuler sur DSP veut dire qu'on ne travaille pas en « temps-réel », donc exécuter nos programmes sur ce dernier demande beaucoup de temps et nécessite plusieurs étapes : éditions de liens, compilation ou assemblage, débogage...etc.
Pour cela, nous avons préféré tester nos algorithmes d'abord sous l'environnement Borland C, afin de nous assurer de leur efficacité, ensuite les transposer sur le DSP. Cette étape nous a permis de récupérer immédiatement l'exécutable de nos programmes et d'effectuer différents tests sur plusieurs types d'images en niveaux de gris, de tailles différentes, pour notre cas nous nous sommes limitées à la taille 128*128 pixels. Les résultats de ces tests seront présentés ultérieurement.

Implémentation sur DSP

Une fois qu'on s'est assuré de l'efficacité de nos programmes écrits en langage C, on a transposé ces derniers sous l'environnement de développement Code Composer Studio (CCS) configuré en simulateur C55x. (voir son fonctionnement en annexe 3).
Le schéma suivant résume le diagramme de développement d'un fichier source écrit en langage C ou en C/C++ sur le DSP :

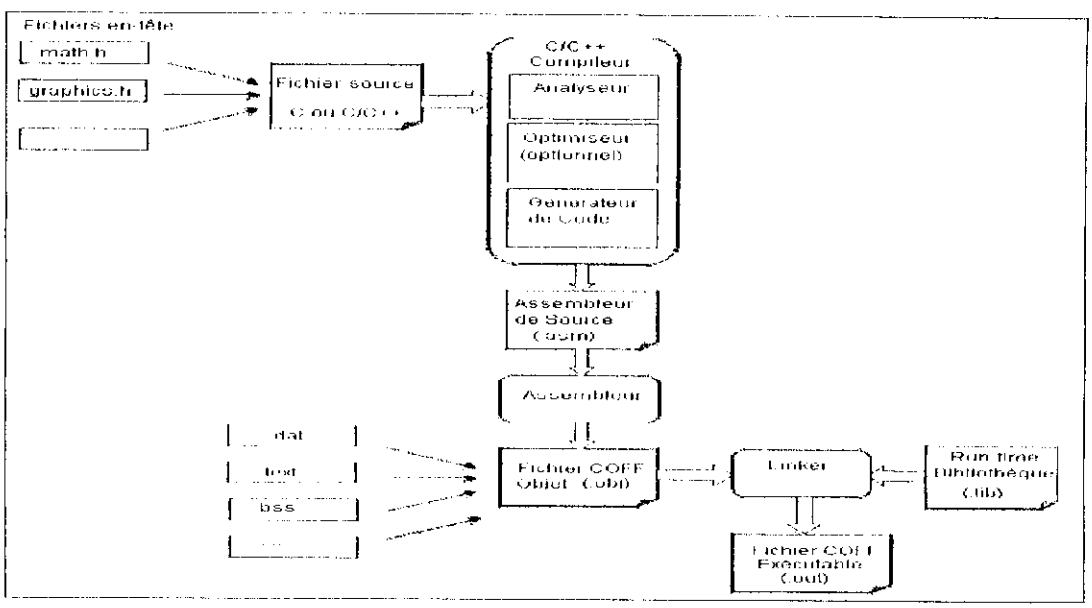


Figure IV.3 : Diagramme de développement d'un fichier source C/C++

IV.2.3.1. Implémentation des filtres optimaux sans les réseaux de neurones

Nous allons présenter dans cette partie, la méthode d'implémentation des filtres optimaux de Shen et de Deriche sur le simulateur Code Composer du DSP, ainsi que les algorithmes associés à ces derniers.

Voici le schéma récapitulatif de notre implémentation :

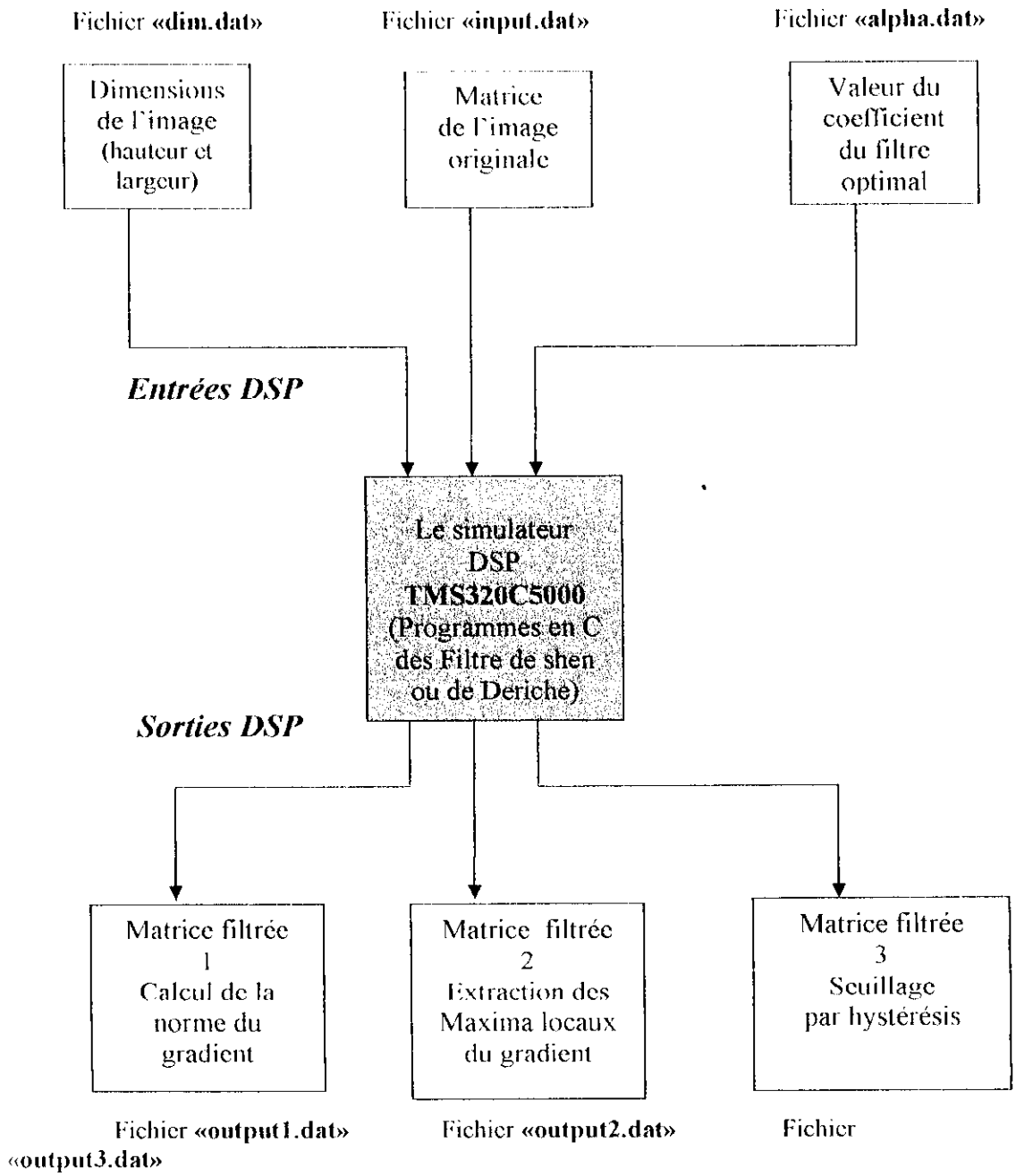


Figure IV.4 : Schéma d'implémentation du filtre optimal sur DSP.

La simulation de nos programmes s'est déroulée comme suit :

- **1ére étape : Création du projet**

Afin d'exécuter notre programme sur le simulateur du Code Composer, nous avons créé un projet (*.pjt), (pour plus de détails sur la façon de créer un projet, voir l'annexe 3), dans lequel nous avons ajouté :

- Le fichier Source (*.c), contenant le code source du filtre optimal ;
- Le fichier de Commande (*.cmd) (linker command file), représentant la configuration de la mémoire de notre système, et qui est nécessaire pour l'édition de liens de notre programme. Ce fichier a été fourni par le simulateur, mais sa configuration mémoire ne nous a pas convenu, elle est insuffisante pour effectuer un traitement de l'image, donc nous l'avons modifié de telle sorte à ce qu'on arrive à traiter des images de taille 128*128 pixels, mais par blocs de 128*8 pixels ;
- Le fichier Bibliothèque (*.lib) (Run-time Library), il est aussi nécessaire pour l'édition de liens de notre programme, il est fourni par le simulateur et doit être utilisé sans aucune modification, on doit juste spécifier son chemin afin de l'ajouter au projet. Pour notre cible, on a utilisé le fichier « rst55.lib ».

- **2ème étape : Introduction des données de l'image à traiter**

Tous les fichiers d'entrées doivent se présenter sous le format (*.dat).

Les fichiers dont on a eu besoin pour notre programme sont :

- Un fichier contenant les dimensions de l'image à traiter, nommé « dim.dat » ;
- Un fichier contenant la matrice de l'image à traiter (les valeurs des niveaux de gris des pixels), nommé « input.dat » ;
- Un fichier contenant la valeur α du filtre de Shen ou de Deriche, nommé « alpha.dat » ;

Remarque : Nous avons obtenu les données de l'image à traiter grâce à une interface réalisée sous l'environnement Builder C++, dont les détails seront présentés au chapitre VI.

- **3ème étape : Compilation et Exécution du programme**

Après avoir créé notre projet et lui avoir introduit toutes les données de l'image à traiter, nous avons procédé à la compilation et l'exécution de notre

programme (Build-Run). Cette étape permet au Code Composer de compiler, assembler et lier notre programme afin de l'exécuter.

- **4ème étape : Récupération des données de l'image filtrée**

Une fois l'exécution accomplie, on obtient en sortie trois fichiers sous format (*.dat) qui représentent les images filtrées finales, que nous avons nommé :

- « output1.dat », correspondant à l'image filtrée après calcul de la norme du gradient ;
- « output2.dat », correspondant à l'image filtrée après extraction des maxima locaux ;
- « output3.dat », correspondant à l'image filtrée après seuillage par hystérésis.

Remarque : Afin de visualiser les matrices filtrées obtenues sous forme d'image, nous avons utilisé notre interface réalisée sous Builder C++ (chapitre VI).

Algorithmes d'implémentation

A. Filtre de Shen

Les filtres de lissage et de dérivation de Shen s'implémentent par filtrage récursif d'ordre 1.

- ✓ Le filtre de lissage : $l(x) = c.e^{-\alpha|x|}$
- ✓ Le filtre de dérivation : $d(x) = \begin{cases} d.e^{-\alpha x} \ldots\ldots si \ldots x \geq 0 \\ d.e^{\alpha x} \ldots\ldots si \ldots x \leq 0 \end{cases}$

B. Filtre de Deriche

Les filtres de lissage et de dérivation de Deriche s'implémentent par filtrage récursif d'ordre 2.

- ✓ Le filtre de lissage : $l(x) = k(\alpha|x| + 1)e^{-\alpha|x|}$
- ✓ Le filtre de dérivation : $d(x) = c.xe^{-\alpha|x|}$

✦ Calcul de la norme du gradient

Le calcul des composantes du gradient s'effectue en calculant les images correspondant aux dérivées partielles des points par rapport à x et à y par filtre séparable récursif :

$$I_x(x,y) = (I * L(y)) * d(x)$$

$$I_y(x,y) = (I^*L(x))^*d(y)$$

Voici l'algorithme correspondant à cette étape :

DEBUT

Lecture des dimensions, de la matrice de l'image à traiter, et du coefficient du filtre optimal

Calcul des coefficients du filtre

Filtrage X

Lecture de la matrice image horizontalement de gauche vers la droite

On transforme la matrice A(N*M) en vecteur unidimensionnel x

Pour i=1,hauteur

Pour j=1,largeur

Faire pour l=1,N (avec N=hauteur*largeur)

x[l]=A[i][j]

Ffaire

Lissage horizontal positif

Pour l=1,N

Faire

Shen yp[l]=a*x[l]+b*yp[l-1]

Deriche yp[l]=a0*x[l]+a1*yp[l-1]-b1yp[l-1]-b2yp[l-2]

Ffaire

Lissage horizontal négatif

Pour l=N,1

Faire

Shen yn[l]=a*x[l+1]+b*yn[l+1]

Deriche yn[l]=a2*x[l+1]+a3*x[l+2]-b1yn[l+1]-b2*yn[l+2]

Ffaire

Lissage final horizontal

Pour l=1,N

Faire ylissx[l]= yp[l] +yn[l] **Ffaire**

Transformation du vecteur lissé en matrice

Pour i=1,hauteur

Pour j=1,largeur

Faire l=1,N Matlissx[i][j] =ylissx[l] **Ffaire**

Lecture de la matrice matlissx verticalement de haut en bas

Pour j=1,largeur

Pour i=1,hauteur

Faire pour l=1,N (avec N=hauteur*largeur)

x[l]=matlissx[i][j]

Ffaire

Derive verticale positive

Pour l=1,N

Faire

Shen yp[l]=d*x[l]+f*yp[l-1]

Deriche yp[l]=a*x[l]-b1*yp[l-1]-b2*yp[l-2]

Ffaire

Derive verticale négative

Pour l=N,1

Faire

Shen yn[l]=-d*x[l]+f*yn[l+1]

Deriche yn[l]=-a*x[l+1]-b1*yn[l+1]-b2yn[l+2]

Ffaire

Derive finale verticale

Pour l=1,N

Faire yx[l]=yp[l]+yn[l] **Ffaire**

Transformation du vecteur lissé en matrice

Pour j=1,largeur

Pour i=1,hauteur

Faire l=1,N

Bx[i][j]=yx[l]

Ffaire

Filtrage Y

Lecture de la matrice image verticalement de droite vers la gauche

On transforme la matrice A(N*M) en vecteur unidimensionnel x

Pour j=1,largeur

Pour i=1,hauteur

Faire pour l=1,N (avec N= hauteur*largeur)

x[l]=A[i][j]

Ffaire

Lissage vertical positif

Pour l=1,N

Faire

Shen yp[l]=a*x[l]+b*yp[l-1]

Deriche yp[l]=a0*x[l]+a1*yp[l-1]-b1yp[l-1]-b2yp[l-2]

Ffaire

Lissage vertical négatif

Pour l=N,1

Faire

Shen yn[l]=a*x[l+1]+b*yn[l+1]

Deriche yn[l]=a2*x[l+1]+a3*x[l+2]-b1yn[l+1]-b2*yn[l+2]

Ffaire

Lissage final vertical

Pour l=1,N

Faire ylissy[l]= yp[l] +yn[l] **Ffaire**

Transformation du vecteur lissé en matrice

Pour j=1,largeur

Pour i=1,hauteur

Faire l=1,N

Matlissy[i][j]=ylissy[l]

Ffaire

Lecture de la matrice matlissy horizontalement de gauche vers la droite

Pour i=1,hauteur

Pour j=1,largeur

Faire pour l=1,N (avec N=hauteur*largeur)

x[l]=matlissy[i][j]

Ffaire

Dérive horizontale positive

Pour l=1,N

Faire

Shen yp[l]=d*x[l]+f*yp[l-1]

Deriche yp[l]=a*x[l]-b1*yp[l-1]-b2*yp[l-2]

Ffaire

```
Dérive horizontale négative
  Pour I=N,1
      Faire
          Shen      yn[I]=-d*x[I]+f*yn[I+1]
          Deriche   yn[I]=-a*x[I+1]-b1*yn[I+1]-b2yn[I+2]
      Ffaire
Dérive finale horizontale
  Pour I=1,N
     Faire        yy[I]= yp[I] +yn[I]        Ffaire
Transformation du vecteur lissé en matrice
    Pour i=1,hauteur
      Pour j=1,largeur
         Faire  I=1,N
                Bx[i][j]=yx[I]
         Ffaire
Calcul de la norme du gradient
    Pour i=1,hauteur
      Pour j=1,largeur
        Faire
            B[i][j]= racine (Bx[i][j]*Bx[i][j]+ By[i][j] +By[i][j])
        Ffaire
```

- **Extraction des maxima locaux**

Après l'étape de dérivation de l'image suivant X (respectivement Y). On applique une extraction des maxima locaux de la norme du gradient. Le but de cette étape est de faire extraire tous les pixels qui représentent un maxima local dans la direction du Gradient. Voici l'algorithme correspondant à cette étape :

```
Pour i=1, hauteur
    Pour j=1, largeur
Faire      Si {Norme_gradient (i,j)> Norme_gradient (in,jn) }Et { Norme_gradient
(i,j)>= Norme_gradient (im,jm) }   // Norme_gradient (in ,jn) et Norme_gradient
(im,jm) représentent les normes des points dans la direction de la norme du gradient //
```

```
            Alors  // prendre le point (i,j) comme contour
                        Image_sortie (i,j) =Norme_gradient (i,j)
            Sinon       // rejeter le point (i,j)          Image_sortie(i,j) = 0
        Fsi
Ffaire
```

- **Seuillage par Hystérésis**

Après l'extraction des maxima locaux de la norme du gradient on applique l'opération de seuillage par hystérésis. Cette étape permet d'éliminer les contours qui ont une faible norme du gradient (distorsion) et de diminuer le nombre de bouts de contour non fermés.

Voici l'algorithme correspondant à cette étape :

```
Pour i=1, hauteur
    Pour j=1, largeur
      Faire        Si Norme_Gradient (i,j)<10  Alors Norme_Gradient(i,j) =0
                     Sinon   Norme_Gradient(i,j) =255
                   Fsi
Ffaire
FIN
```

Pour faire face à la contrainte : « insuffisance de la mémoire » rencontrée sur le DSP, il a fallu boucler cette algorithme 16 fois (blocs de 128*8 pixels) afin de traiter toute l'image (128*128 pixels).

IV.2.3.2. Implémentation des filtres optimaux en utilisant les réseaux de neurones

L'une des propriétés les plus importantes des réseaux de neurones est leur capacité de généralisation, qui leur permet de donner des réponses satisfaisantes aux exemples qui ne font pas partie de leur ensemble d'apprentissage.

Nous allons présenter dans cette partie, la méthode d'implémentation des filtres optimaux de Shen et de Deriche par les réseaux de neurones sur le simulateur Code Composer du DSP.

Pour notre système nous avons utilisé un réseau multicouche perceptron qui est le mieux adapté pour le filtrage optimal.

↓ L'opérateur Neuronal : « Filtre optimal »

On définit l'opération du Filtre neuronal Ψ sur une image I(x,y) :

$$\Psi = h \circ k \circ h^{-1}$$

Tel que : Ψ (I(x,y)) = Filtre Neuronal (I(x,y)).

h : est une fonction qui change l'échelle sur les données d'entrées pour qu'elles puissent être utilisables par le réseau (données comprises entre 0 et 1), l'entrée du réseau est ainsi obtenue par la relation : $f = h(I(x,y))$

k : la fonction qui calcule la sortie du réseau en fonction de o tel que $o = k(f)$.

h^{-1} : La fonction inverse de la fonction h, son rôle est de convertir la sortie du réseau en valeurs en niveaux de gris pour l'image résultat.

$$h^{-1}(h(I(x,y))) = I(x,y).$$

Pour le traitement d'image à 256 niveaux de gris, la fonction h est de la forme :

$h = 1/255$ et la fonction h^{-1} est alors : $h^{-1} = 255$.

↓ Architecture du réseau

A. Principe de fonctionnement

L'architecture représentée (Figure IV.5) est celle d'un filtre neuronal pour lequel la fenêtre sur l'image est de taille 3*3. Cette taille peut varier suivant le filtre que l'on veut réaliser, donc suivant le nombre de pixels du voisinage qui va être pris en compte.

Les filtres qu'on veut réaliser sont les filtres optimaux de Shen-Castan et de Deriche, pour cela le réseau utilisé est un réseau perceptron multicouche composé de:

1. Une couche d'entrée à 9 neurones destinée à recevoir le pixel à traiter avec ses 8 voisins connexes ;
2. Une couche cachée de 9 neurones (nombre choisi après plusieurs essais).
3. Une couche de sortie à 1 neurone, qui représente le pixel filtré.

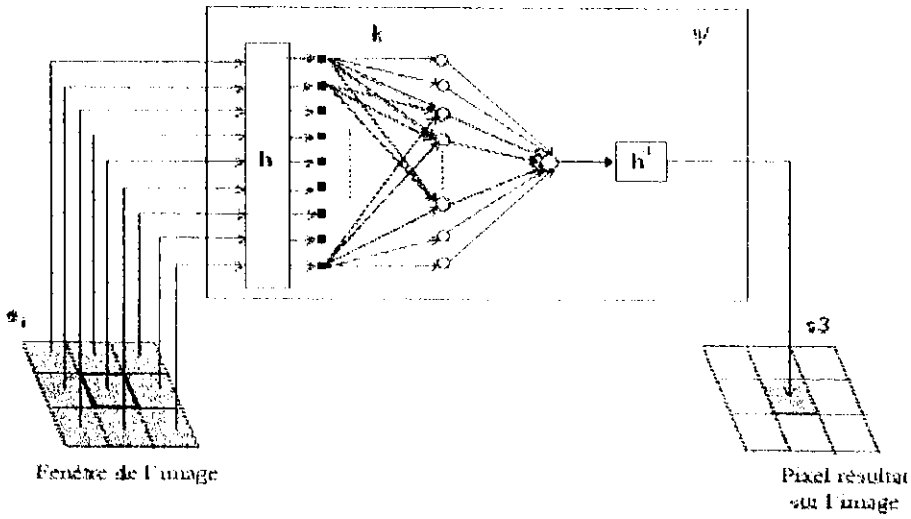


Figure IV.5 : Architecture et fonctionnement du réseau neuronal.

B. Parcours de l'image par le filtre neuronal

Lors de l'apprentissage et de la phase de filtrage d'une image, la fenêtre de balayage du filtre neuronal (masque =3*3) est déplacée pixel par pixel sur l'image, cette fenêtre doit être identique pour la totalité de l'image.

Les bords de l'image ne peuvent pas être traités par le filtre neuronal, donc ils sont mis à zéro.

Cette architecture permet de filtrer des images de tailles différentes, les images ne nécessitent donc pas de phase de normalisation avant la phase de filtrage. Afin de pouvoir implémenter ce filtre sur le DSP nous nous sommes limitées au traitement des images de taille 128*128 pixels.

La figure IV.6 présente le parcours de l'image par le réseau de neurones :

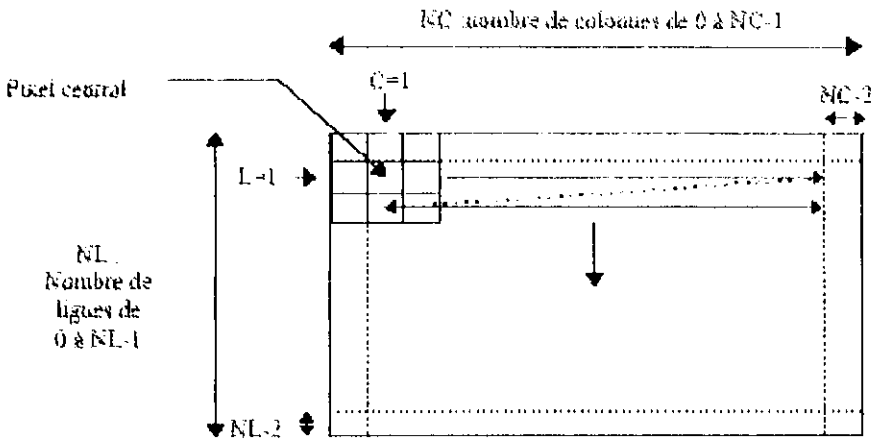


Figure IV.6 : Parcours de l'image.

✦ Apprentissage du réseau

Avant de pouvoir utiliser notre réseau, on doit d'abord déterminer les valeurs des poids de ses connexions. Ces valeurs sont déterminées durant la phase d'apprentissage.

Pour l'apprentissage de notre perceptron multicouches on utilise la règle de rétropropagation du gradient. Cette règle permet de déterminer les valeurs des poids dans toutes les couches du réseau en fonction de l'erreur entre la valeur obtenue et la valeur désirée.

Pour que le réseau puisse apprendre une fonction désirée, il faut lui présenter une série d'exemples appropriés. Dans notre cas, ces exemples seront issus d'images puisque notre réseau de neurones doit apprendre à filtrer des images.

Nous avons utilisé MATLAB 6.5 pour ce calcul, et nous avons sauvegardé ces résultats dans un fichier pour l'implémentation de nos filtres.

L'étape d'apprentissage s'effectue comme suit :

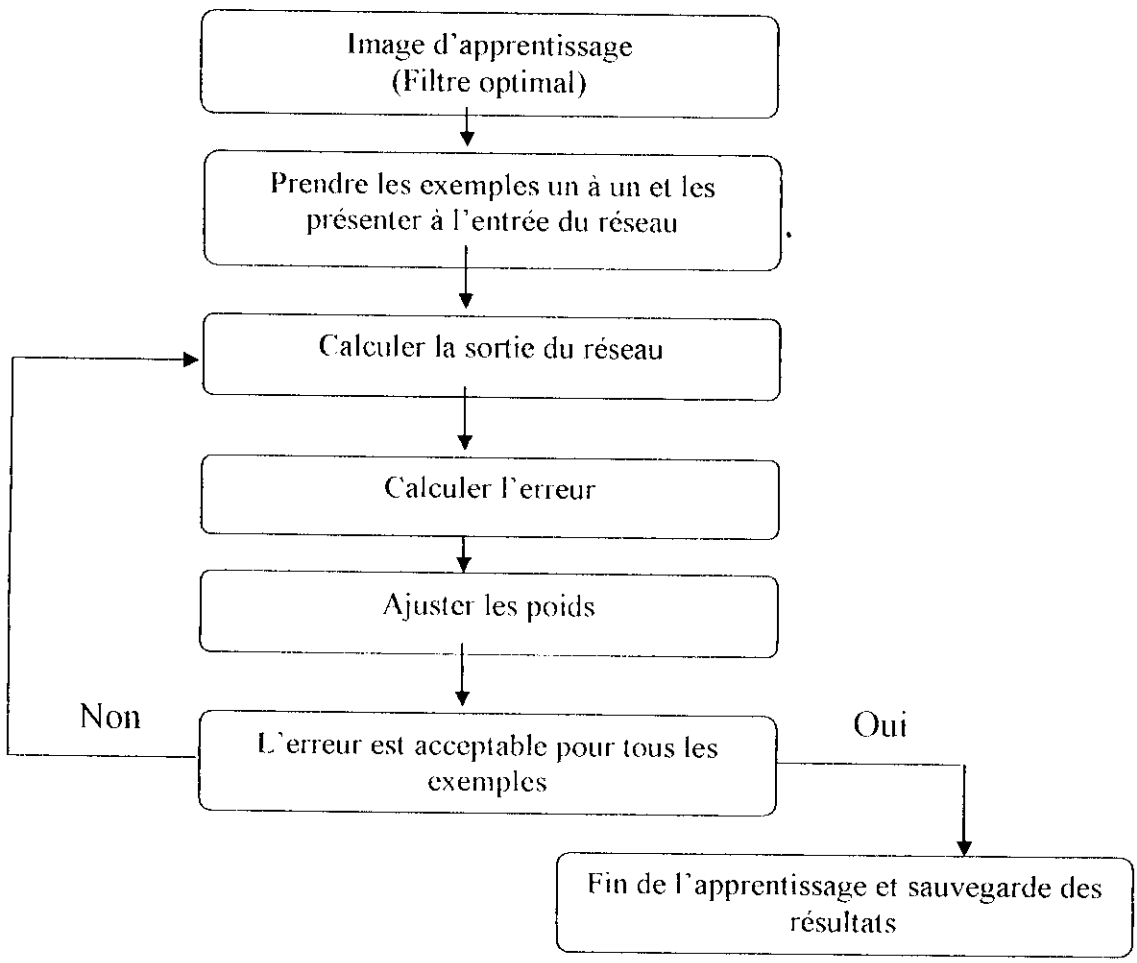


Figure IV.7 : Etapes d'apprentissage.

Plusieurs tests d'apprentissage ont été faits concernant le filtre de Shen-Castan et celui de Deriche, dont les images d'apprentissage sont celles obtenues par le calcul de la norme du gradient.

Le tableau suivant résume les meilleurs résultats obtenus lors de ces apprentissages.

Filtres optimaux	Fonctions d'activation			Nombre d'itérations	Temps d'apprentissage	Erreur commise (MSE)
	Couche 1	Couche 2	Couche 3			
Shen-Castan	sigmoïde	sigmoïde	Identité	500	1 h:04	0.00386738
Deriche	sigmoïde	Sigmoïde	Identité	200	30 minutes	0.0292296

Tableau IV.1 : Résultats d'apprentissage.

Lors de l'apprentissage, on obtient les courbes d'erreur suivantes :

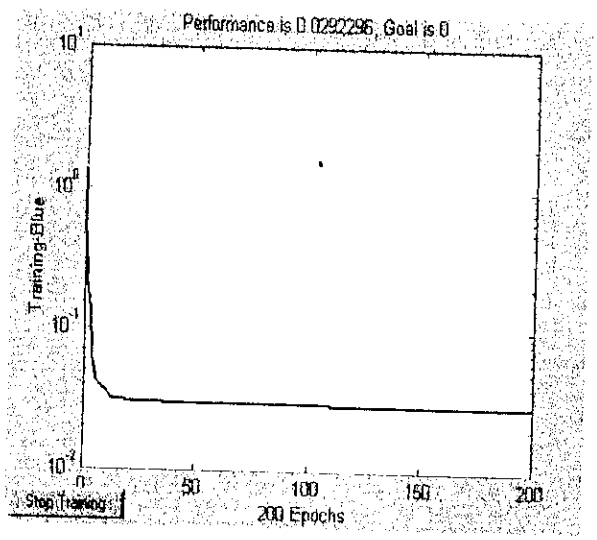


Figure IV.8 : Erreur d'apprentissage pour Deriche.

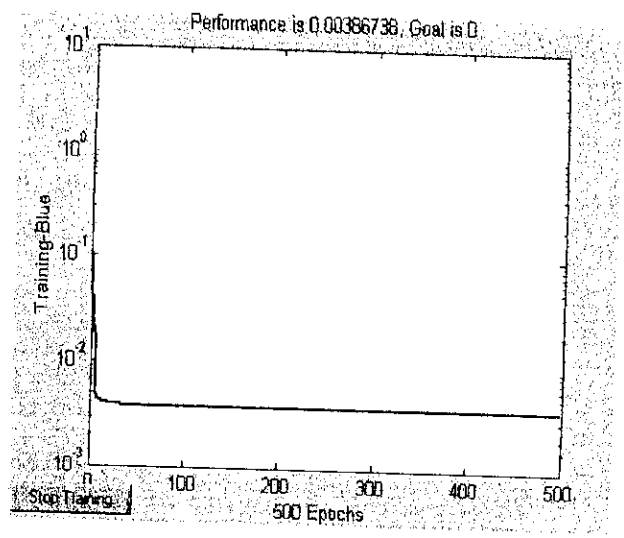


Figure IV.9 : Erreur d'apprentissage pour Shen-Castan.

On remarque que l'erreur d'apprentissage de Shen-Castan est inférieure à celle de Deriche, cela explique que le filtre de Shen-Castan a mieux appris la fonction du filtre optimal.

➧ Implémentation du réseau sur DSP

La figure IV.10 donne un schéma récapitulatif de notre implémentation :

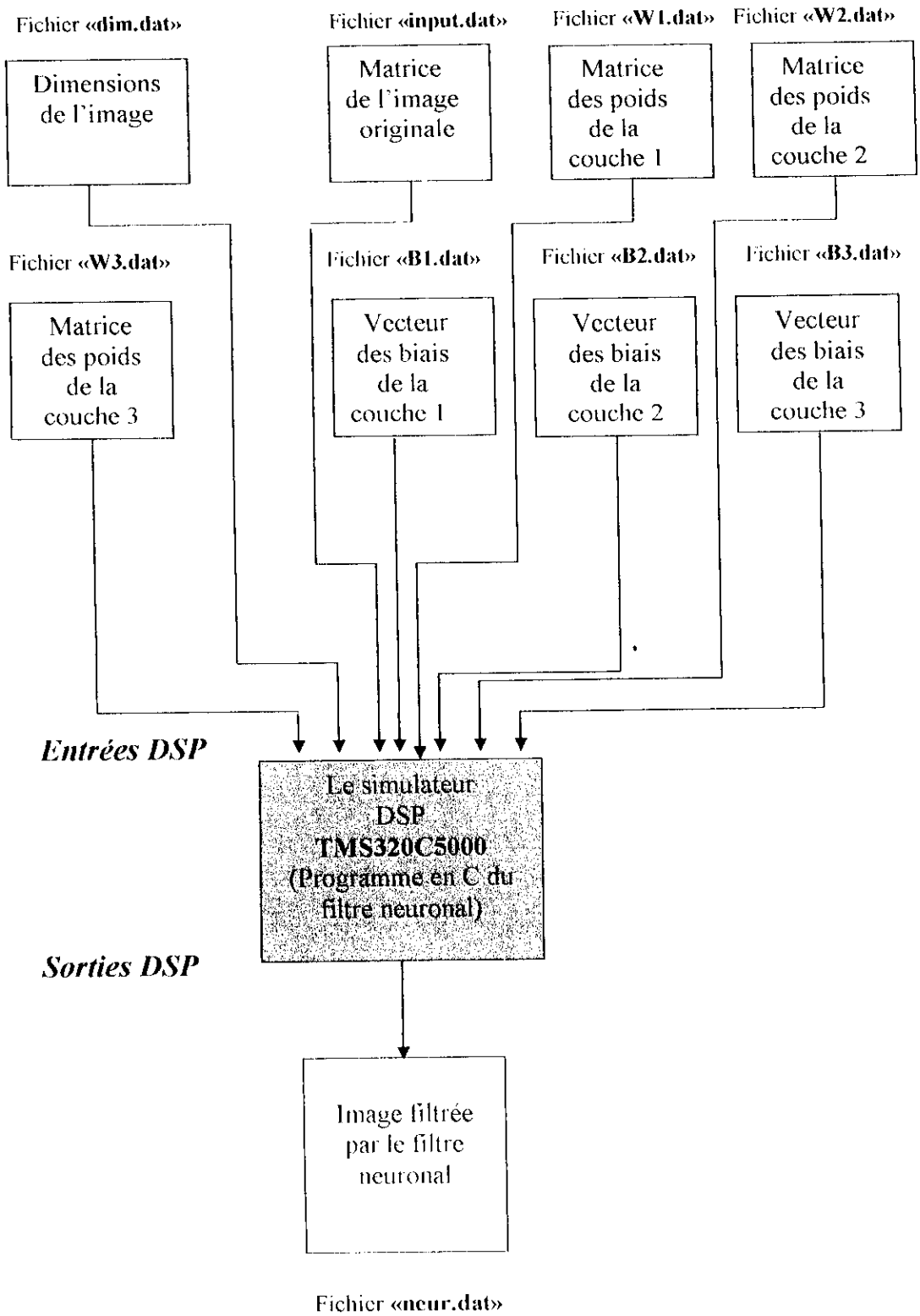


Figure IV.10 : Schéma d'implémentation du filtre neuronal sur DSP.

La simulation de nos programmes s'est déroulée comme suit :

- **1ére étape : Création du projet**

Afin d'exécuter notre programme sur le simulateur du Code Composer, on a créé un projet (*.pjt), dans lequel on a ajouté :

- Le fichier Source (*.c), contenant le code source du filtre neuronal ;
- Le fichier de Commande (*.cmd) (linker command file), modifié de telle sorte à ce qu'on arrive à traiter des images de taille **128*128** pixels.
- Le fichier Bibliothèque (*.lib) (Run-time Library), pour notre cible, on a utilisé le fichier « rst55.lib ».
- Or la configuration mémoire du DSP qu'on a établi nous a limité à traiter les images par blocs de 128*8 pixels. Cette contrainte peut être soulevée en trouvant une configuration mémoire convenable pour traiter des images de taille plus grande et sans blocs.

- **2ème étape : Introduction des données**

Les fichiers (*.dat) dont on a eu besoin pour notre programme sont :

- Un fichier contenant les dimensions de l'image à traiter, nommé « dim.dat » ;
- Un fichier contenant la matrice de l'image à traiter (les valeurs des niveaux de gris des pixels), nommé « input.dat » ;
- Les six fichiers contenant les valeurs des poids et biais obtenus lors de l'apprentissage du filtre de Shen ou de Deriche, nommés respectivement «W1.dat », « W2.dat », « W3.dat », « B1.dat », « B2.dat », et « B3.dat » ;

Les fichiers dimensions et matrice de l'image à traiter sont obtenus grâce à notre interface, et les poids et les biais grâce à Matlab.

- **3ème étape : Compilation et Exécution du programme**

Après avoir créé notre projet et lui avoir introduit toutes les données nécessaires pour le filtrage neuronal, on a procédé à la compilation et l'exécution de notre programme (Build-Run).

- **4ème étape : Récupération des données de l'image filtrée**

Après que notre exécution soit accomplie, on obtient en sortie un fichier sous format (*.dat) qui représente l'image filtrée finale correspondant au filtre optimal (Shen ou Deriche) appris par le réseau neuronal, qu'on a nommé : «neur.dat».

De même, afin de visualiser cette matrice filtrée obtenue sous forme d'image, nous avons utilisé notre interface.

Algorithmes d'implémentation

```
DEBUT
Lecture des poids, des biais, des dimensions, et de la matrice de l'image à traiter
Pour i=2, hauteur-1
  Pour j=2, largeur-1
Faire
 Création de la cellule 9*9 d'entrée de la couche 1
      e[1]=A[i-1][j-1] ; e[2]=A[i-1][j] ; e[3]=A[i-1][j+1] ; e[4]=A[i][j-1] ; e[5]=A[i][j];
      e[6]=A[i][j+1] ; e[7]=A[i+1][j-1] ; e[8]=A[i+1][j] ; e[9]=A[i+1][j+1];
 Calcul de la sortie de la couche 1
Pour  n=1, 9
     Faire     s1[n]=0
          Pour p=1 ,9   Faire s1[n]=s1[n]+(w1[n][p]*e[p])  Ffaire
               s1[n]=1/(1+exp(-(s1[n]+b1[n])))
Ffaire
Calcul de la sortie de la couche 2
Pour n=1, 9
 Faire
   s2[n]=0
      Pour p=1 ,9   Faire s2[n]=s2[n]+(w2[n][p]*s1[p])  Ffaire
   s2[n]=1/(1+exp(-(s2[n]+b2[n])))
Ffaire
Calcul de la sortie couche 3
s3=0
Pour n=1, 9  Faire s3=s3+w3[n]*s2[n]  Ffaire
 s3= 255*s3+b3
 Ffaire
FIN
```

IV.2.4. Comparaisons

Dans cette partie on compare les images obtenues par le filtre optimal sans l'utilisation des réseaux de neurones avec celles filtrées par le filtre optimal en utilisant les réseaux de neurones. Le schéma suivant illustre la méthode de comparaison

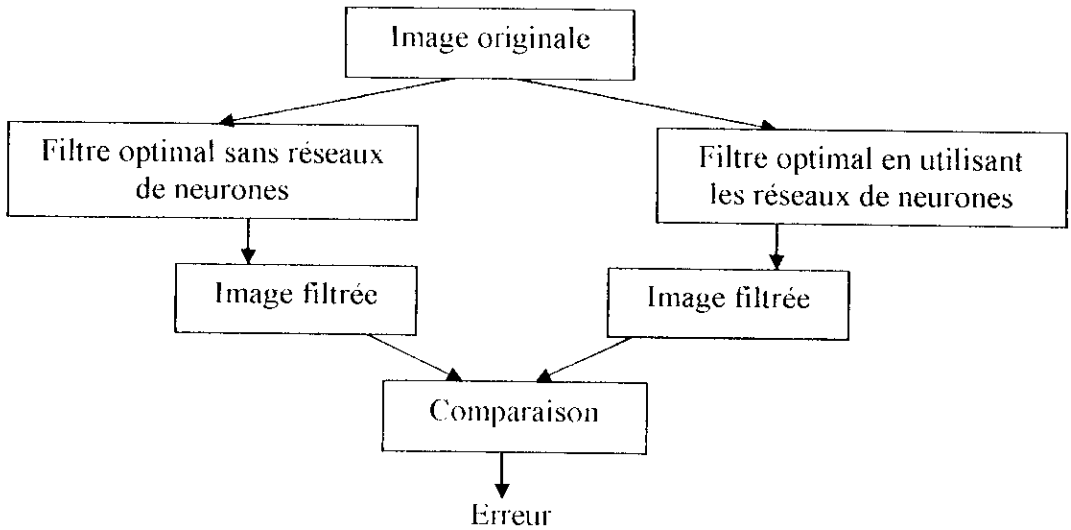


Figure VI.11 : Méthode de comparaison.

Pour pouvoir vérifier les résultats obtenus, il faut trouver un moyen autre que le critère visuel, qui n'est pas très objectif, permettant de comparer les deux images.

La méthode qui a été utilisée est celle de la mesure de l'erreur moyenne quadratique (Mean Square Error, MSE).

Cette méthode consiste à effectuer une différence entre les pixels des deux images à comparer. Le calcul se fait avec la formule suivante :

$$MSE = \frac{1}{M * N} \sum_{i=1}^{M*N} (x_i - y_i)^2 \quad \ldots\ldots(38)$$

Où x_i et y_i représentent les pixels normalisés des images à comparer, M et N étant les dimensions de l'image.

Algorithme de comparaison

DEBUT

Lecture des deux matrices images X, Y à comparer

MSE=0

```
Pour i=1,hauteur
  Pour j=1,largeur
   Faire
     MSE =MSE+(X[i][j]-Y[i][j])* (X[i][j]-Y[i][j])
   Ffaire
MSE=MSE/(hauteur*largeur*255*255)
FIN
```

IV.3. Conclusion

Dans ce chapitre nous avons décrit les différentes méthodes, outils et algorithmes de notre travail, qui a consisté en :

- Un filtrage optimal : filtre détecteur de contours de Shen-Castan et celui de Deriche , dont les algorithmes ont d'abord été testés sur le Borland C afin de s'assurer de leurs efficacité, puis transposés sur le simulateur Code Composer du DSP ;
- Un filtrage neuronal : notre réseau de neurone a appris la fonction du filtre optimal de Shen-Castan ainsi que celle de Deriche. L'étape d'apprentissage a été faite sur Matlab, ensuite l'algorithme du filtre neuronal a aussi été testé sur Borland C puis transposé sur le simulateur Code Composer du DSP.

La configuration mémoire du DSP que nous avons établi nous a contraint à traiter les images par blocs de 128*8 pixels pour le filtrage optimal et de 128*128 pixels pour le filtrage neuronal. Cette contrainte peut être soulevée en trouvant une configuration mémoire convenable pour traiter des images de taille plus grande et sans blocs.
Le chapitre qui suit sera consacré aux résultats et interprétations de nos filtres.

V.1. Introduction

Dans cette partie, nous allons présenter les différents résultats des filtres optimaux de Shen-Castan et de Deriche, sans et avec utilisation des réseaux de neurones. Ces filtres ont été effectués sur plusieurs types d'images en niveaux de gris de taille 128*128 pixels.
Nous allons évaluer les résultats visuellement (la continuité, la détection et l'épaisseur des contours), et nous donnerons le temps de traitement pour chaque opération.
La comparaison entre le filtrage optimal sans et avec utilisation des réseaux de neurones se fera par calcul de l'erreur (MSE).
Nous avons effectué plusieurs tests pour plusieurs valeurs de alpha, les meilleurs résultats ont été obtenus pour les valeurs suivantes :

- Shen-Castan : $\alpha = 0.5$;
- Deriche : $\alpha = 1$.

Nous allons présenter deux exemples sélectionnés parmi l'ensemble des images traitées. Ces exemples sont :

- Une image en niveaux de gris représentant un *bureau* ;
- Une image en couleur,d'un *chromosome* que nous transformerons en niveaux de gris avant de la traiter.

V.2. Tests sur le filtrage optimal sans réseaux de neurones

Nous allons présenter les différents résultats des filtres testés sur le C Borland et sur le DSP

V.2.1. Résultats obtenus sur Borland C

- **Filtre de Shen-Castan**

Figure V.1
Image originale

Figure V.2
Norme du gradient

Figure V.3
Extraction des extrema locaux

Figure V.4
Seuillage par Hystérésis

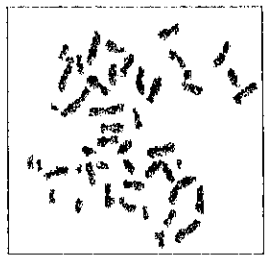


Figure V.5
Image originale

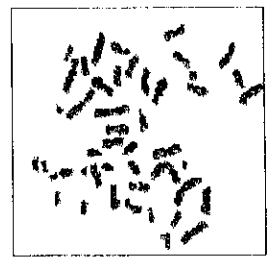


Figure V.6
Image en Niveaux de Gris

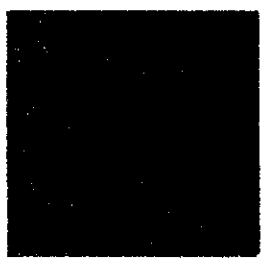


Figure V.7
Norme du gradient

Figure V.8
Extraction des extrema locaux

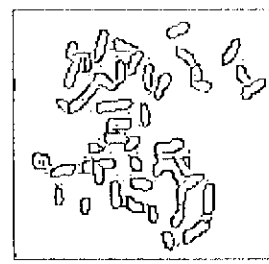


Figure V.9
Seuillage par Hystérésis

- **Filtre de Deriche**

Figure V.10
Image originale

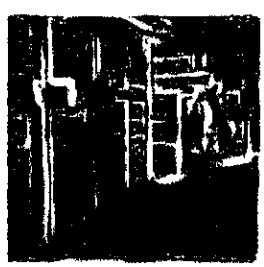


Figure V.11
Norme du gradient

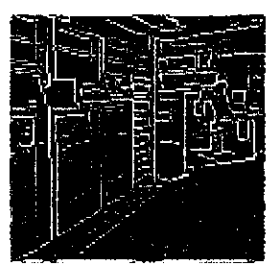


Figure V.12
Extraction des extrema locaux

Figure V.13
Seuillage par Hystérésis

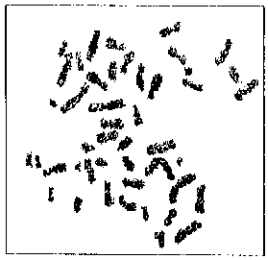


Figure V.14
Image originale

Figure V.15
Image en Niveaux de Gris

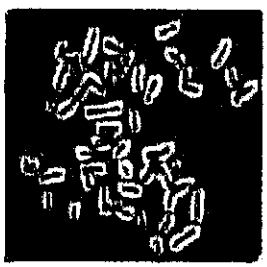


Figure V.16
Norme du gradient

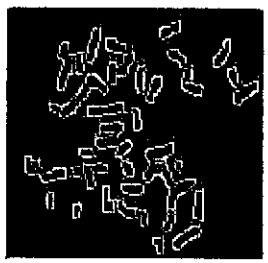


Figure V.17
Extraction des extrema locaux

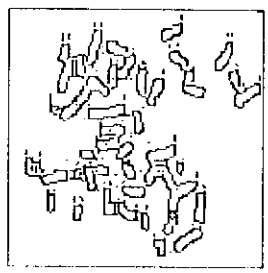


Figure V.18
Seuillage par Hystérésis

- **Interprétations**

Pour les deux filtres, nous remarquons que les variations des niveaux de gris pour les images filtrées par le calcul de la norme du gradient donnent des gradients élevés et donc des traits clairs (plus les pixels sont clairs et plus le gradient a une valeur élevée). Comme nous pouvons le remarquer, les traits sont flous ; c'est pour cela que l'on applique une recherche des extrema locaux et un seuillage par hystérésis, afin de raffiner le contour.
En comparant les figures V.4 et V.9 avec V.13 et V18, nous remarquons que le détecteur de contours de Deriche présente de meilleurs résultats que celui de Shen-Castan. Avec un seuillage par hystérésis de seuil =10, le détecteur de Deriche donne plus de détails sur les contours.

Le temps de traitement de cette opération a duré quelques secondes.

V.2.2. Résultats obtenus sur DSP

- **Filtre de Shen-Castan**

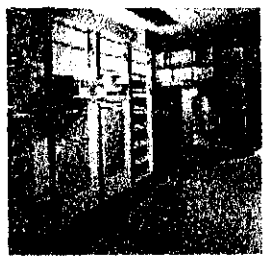


Figure V.19
Image originale

Figure V.20
Norme du gradient

Figure V.21
Extraction des extrema locaux

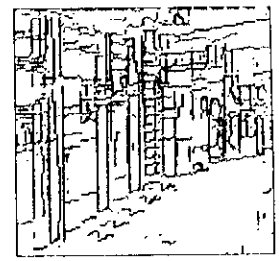


Figure V.22
Seuillage par Hystérésis

Figure V.23
Image originale

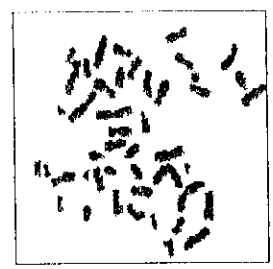


Figure V.24
Image en Niveaux de Gris

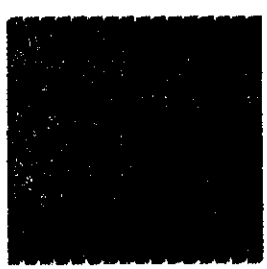


Figure V.25
Norme du gradient

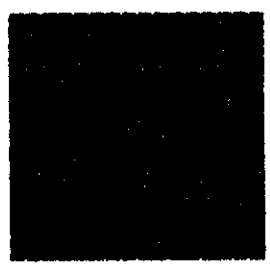


Figure V.26
Extraction des extrema locaux

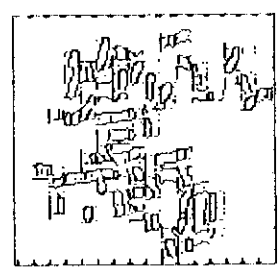


Figure V.27
Seuillage par Hystérésis

- **Filtre de Deriche**

Figure V.28
Image originale

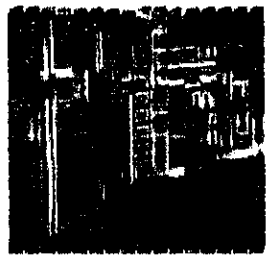


Figure V.29
Norme du gradient

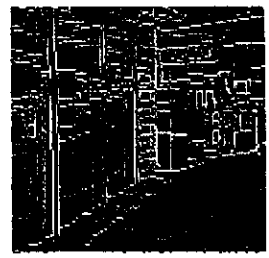


Figure V.30
Extraction des extrema locaux

Figure V.31
Seuillage par Hystérésis

Figure V.32
Image originale

Figure V.33
Image en Niveaux de Gris

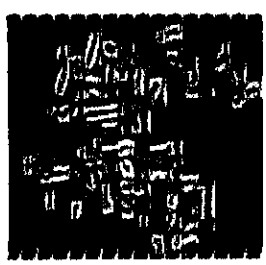


Figure V.34
Norme du gradient

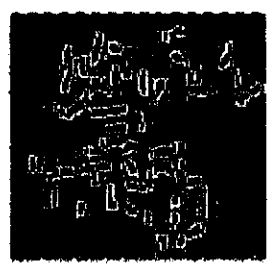


Figure V.35
Extraction des extrema locaux

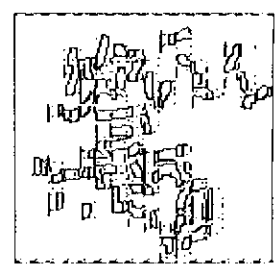


Figure V.36
Seuillage par Hystérésis

- **Interprétations**

Les mêmes remarques que précédemment sont notées ici, sauf que dans ce cas les images ont été traitées par blocs, ceci est très visible car on observe clairement les discontinuités et les effets de bords sur nos images filtrées. Nous avons essayé de remédier à ce problème en commençant le traitement de chaque bloc à partir des deux colonnes du bloc précèdent sans succès (le résultat reste inchangé).

Le temps d'exécution de cette opération sur le simulateur Code Composer du DSP a duré 5 heures.

V.3. Tests sur le filtrage optimal en utilisant les réseaux de neurones sur DSP

- **Filtre de Shen-Castan**

Figure V.37
Image originale

Figure V.38
Filtrage neuronal

Figure V.39
Image originale

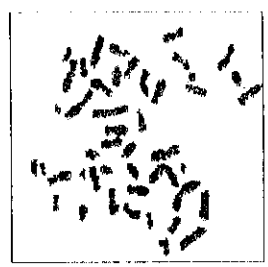


Figure V.40
Image en Niveaux de Gris

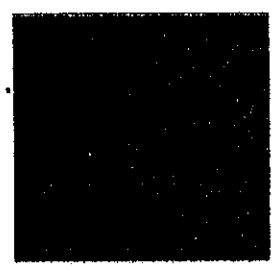


Figure V.41
Filtrage neuronal

- **Filtre de Deriche**

Figure V.42
Image originale

Figure V.43
Filtre neuronal

Figure V.44
Image originale

Figure V.45
Image en Niveaux de Gris

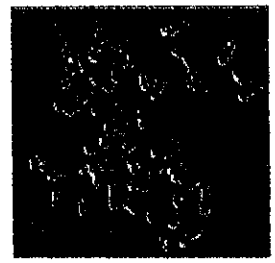


Figure V.46
Filtrage neuronal

- **Interprétations**

Rappelons que les images utilisées pour l'apprentissage de nos réseaux neuronaux ont été celles obtenues par le calcul de la norme du gradient, et que nos réseaux ont appris les fonctions du filtrage optimal.

De même pour le filtrage neuronal, le filtre de Deriche donne de meilleurs contours par rapport au filtre de Shen-Castan.

Nous remarquons que le filtre neuronal donne plus de détails que le filtre optimal non neuronal avec des contours moins flous et plus minces.

Le temps d'exécution de cette opération sur le simulateur Code Composer du DSP a été très long, il a duré 30 heures. Ceci étant justifié par le fait que le filtre neuronal opère par cellule de neuf pixels alors que le filtre non neuronal travaille au niveau du pixel.

- **Comparaisons par calcul d'erreurs**

Nous avons comparé les images obtenues par le filtrage optimal neuronal avec celles du filtrage optimal non neuronal par calcul de l'erreur quadratique moyenne (MSE).

Nous avons obtenu les valeurs suivantes

Filtre de Shen-Castan	Image_Bureau	MSE= 0.008117
	Image_Chromosome	MSE= 0.014087
Filtre de Deriche	Image_Bureau	MSE= 0.057628
	Image_Chromosome	MSE= 0.055498

Nous remarquons que le filtre de Shen-Castan donne une erreur plus faible que celui de Deriche,

V.4. Conclusion

Dans ce chapitre nous avons présenté les résultats des deux techniques de filtrages optimaux non neuronaux et de filtrages optimaux neuronaux. Pour les deux techniques, le filtre de Deriche s'est avéré meilleur que celui de Shen-Castan.

En comparant ces deux techniques, la seconde a donné de meilleurs contours et nous a été plus simple à implémenter sur le DSP, car elle nous a permis de traiter la totalité de l'image, par contre le temps d'exécution a été beaucoup plus long.

Il est à noter que l'utilisation des réseaux de neurones a effectué en quelque sorte une optimisation (ou plus précisément une amélioration) du filtre optimal du fait que les détails apparaissent mieux et que les contours sont moins flous et plus fins.

VI.1. Introduction

Dans ce chapitre nous allons présenter l'interface que nous avons réalisée.
L'expansion de l'environnement Windows a offert à l'utilisateur un ensemble d'outils performant et facile à l'emploi.
Notre interface a été réalisée grâce à l'outil de programmation Builder C++ version 5 sous environnement Windows XP. Le choix du langage est justifié par la souplesse et la simplicité avec laquelle celui-là supporte la programmation orientée objet, et par le besoin d'un environnement de programmation évolué, permettant une description rapide des algorithmes en POO (Programmation Orientée Objet).
C'est une application MDI (Multiple Document Interface) offrant ainsi à l'utilisateur la possibilité de visualiser plusieurs images en même temps. La présence d'un menu bien aménagé, et d'une barre à outils, facilite l'utilisation de l'interface.

Cette interface nous a été utile pour la création et la visualisation des données de l'image pour effectuer notre traitement sur le DSP.
En effet, après l'étape de niveaux de gris, nous avons récupéré l'image à traiter sous forme de matrice ainsi que les dimensions de cette dernière, qui nous ont servi comme fichiers d'entrée pour le DSP sous le format (*.dat).
La valeur alpha du filtrage optimal sera aussi récupérée sous forme de fichier d'entrée pour le DSP sous le format (*.dat).
Une fois le traitement accompli sur le DSP, on récupère les matrices filtrées par les deux filtres optimaux Shen-Castan et Deriche pour les deux techniques de filtrage choisies (filtrage optimal sans et avec l'utilisation des réseaux de neurones) sous le format (*.dat) et nous les visualisons sous forme d'images.

Nous avons ajouté quelques traitements supplémentaires implémentés sous Builder C++ comme la binarisation et l'histogramme en niveaux de gris.

Dans ce qui suit nous présenterons la fenêtre principale de l'application et les menus secondaires avec leurs différentes opérations.

VI.2. Configuration de l'application

L'interface est représentée ci-dessous par la figure suivante :

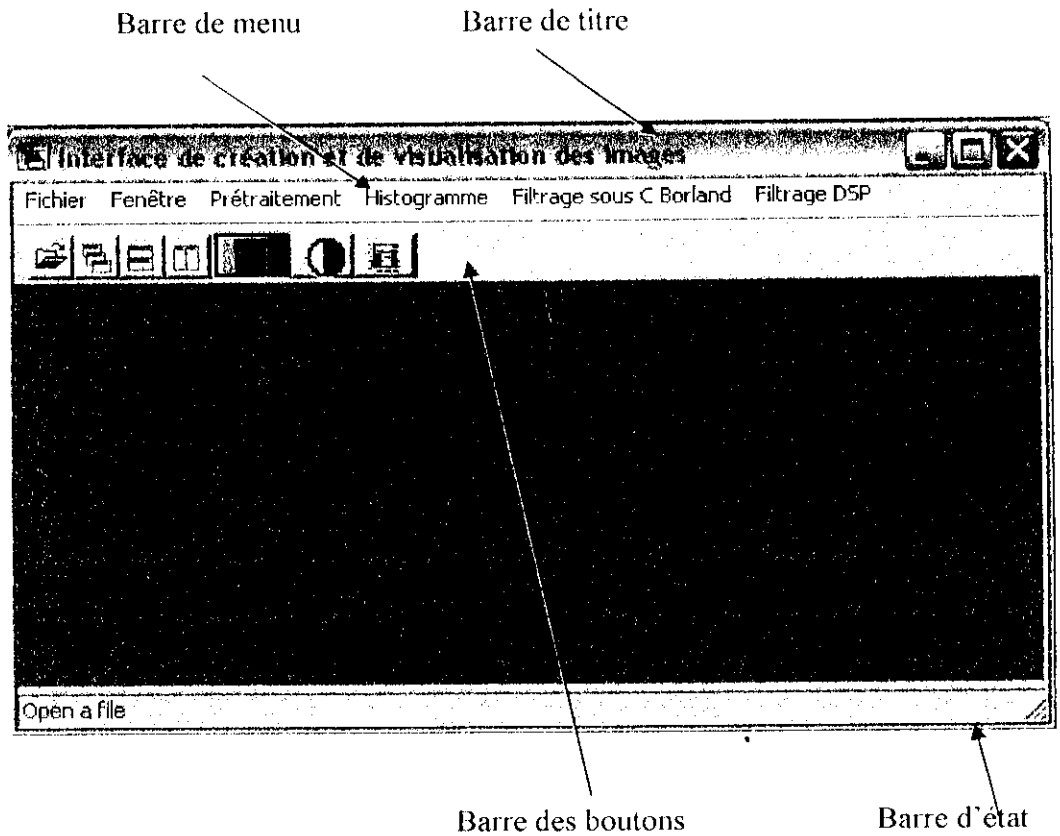


Figure VI.1 : Fenêtre principale.

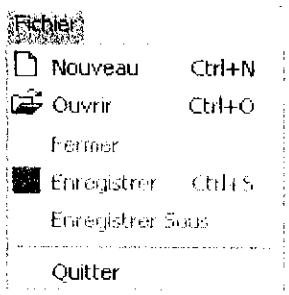
VI.2.1. Fenêtre enfant

Cette application est basée sur le modèle MDI ; ceci veut dire que chaque traitement se fait par rapport à l'image d'origine et s'affiche dans une nouvelle fenêtre enfant.

VI.2.2. La barre des menus

C'est une barre contenant des articles qu'on peut sélectionner à l'aide de la souris. L'activation d'un article fait appel soit à une fonction, soit à une boîte de dialogue.

VI.2.2.1 Menu Fichier



- **Commande ouvrir**

A l'exécution de cette commande une boîte de sélection apparaît permettant à l'utilisateur de choisir le nom du fichier image, d'extension (*.bmp), à ouvrir.

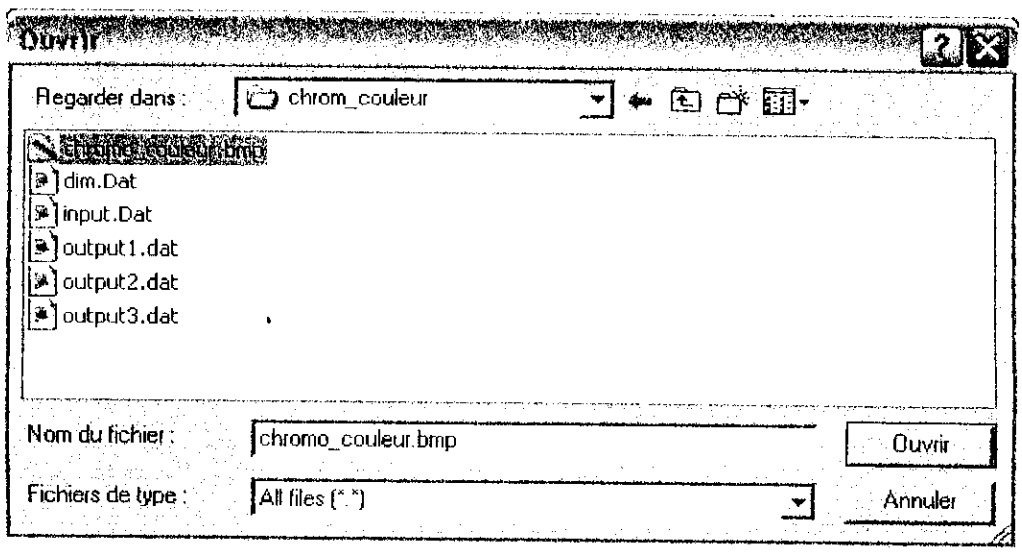


Figure VI.2 : Boîte de dialogue d'ouverture d'images.

- **Commande Enregistrer Sous**

A l'exécution de cette commande une boîte de sélection apparaît permettant à l'utilisateur de sauvegarder l'image qui se trouve dans la fenêtre active, sous un autre nom de fichier.

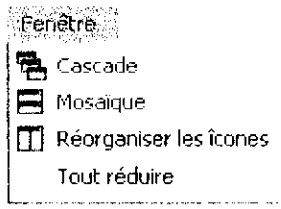
- **Commande Enregistrer**

A l'exécution de cette commande une boîte de sélection apparaît permettant à l'utilisateur de sauvegarder l'image qui se trouve dans la fenêtre active.

- **Commande Quitter**

Cette commande permet de mettre fin à l'application.

VI.2.2.2 Menu Fenêtre



Grâce à ce menu, on peut choisir la manière d'afficher nos différentes fenêtres en cascade, en mosaïque, les réorganiser ou les réduire.

VI.2.2.3 Menu Prétraitement

- **Commande Niveaux de gris**

Après avoir ouvert notre image, qui peut être en couleur, on lui applique cette commande pour l'avoir en niveau de gris. Cette étape est nécessaire pour notre filtrage optimal.

- **Commande Binarisation**

La binarisation est l'étape qui consiste à transformer l'image en seulement deux niveaux de gris le blanc et le noir et ceci se fait par un calcul automatique du seuil ou bien manuellement. Le choix de la méthode de calcul se fait grâce à la boîte de dialogue présentée ci-dessous :

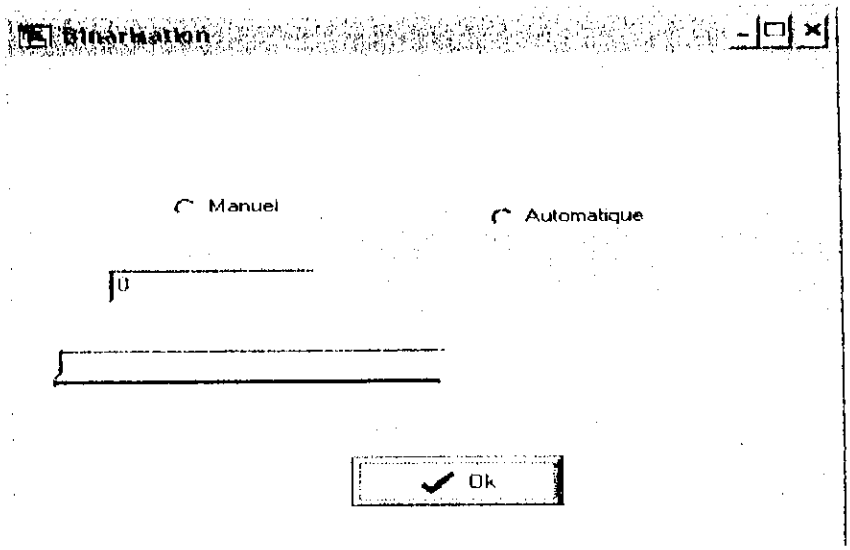
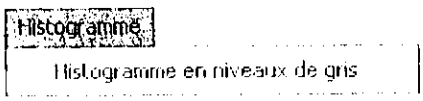


Figure VI.3 : Boîte de dialogue d'introduction du seuil de binarisation.

VI.2.2.4 Menu Histogramme



Dans ce menu, nous ferons l'histogramme en niveaux de gris.

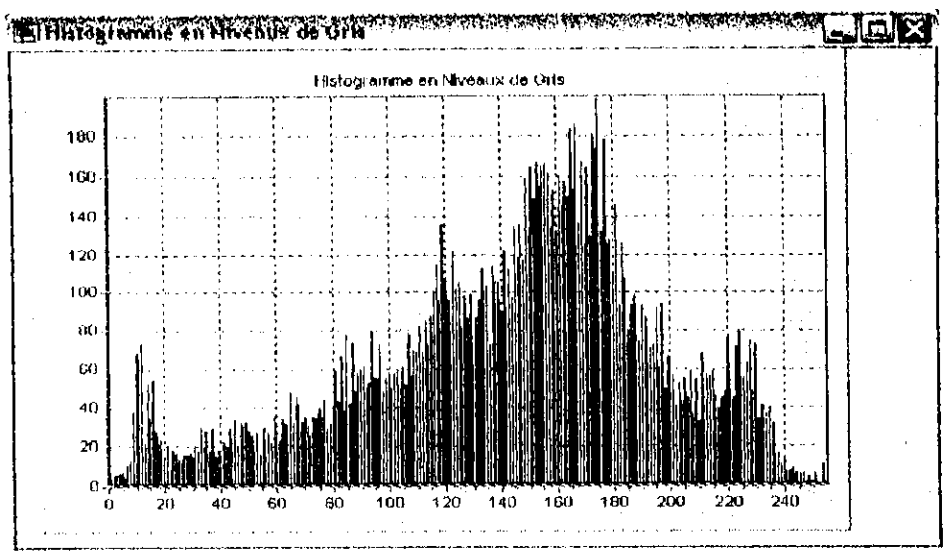


Figure VI.4 : Représentation de l'histogramme en niveaux de gris.

VI.2.2.5 Menu Valeur de alpha

Ce menu sert à introduire et à enregistrer la valeur alpha du filtre optimal de Shen-Castan ou de Deriche dans un fichier qui servira par la suite comme entrée pour le DSP.

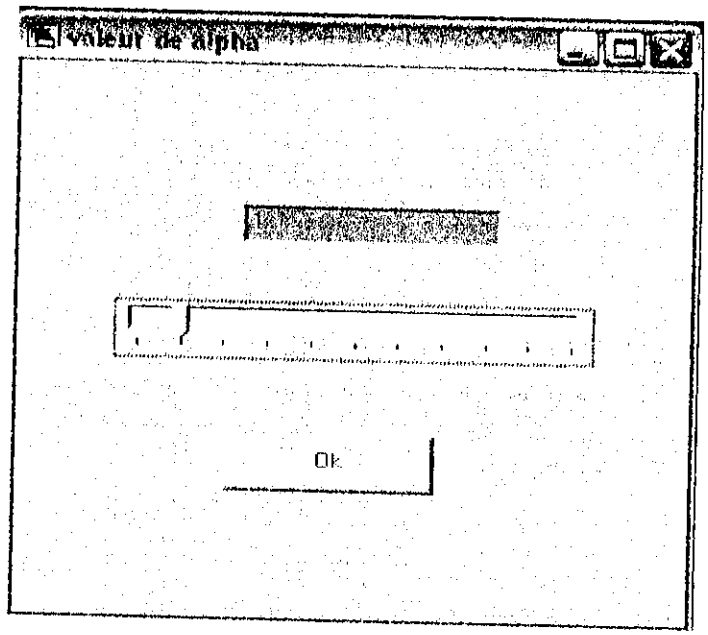
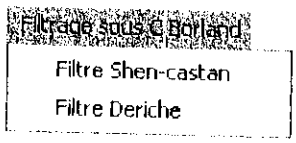


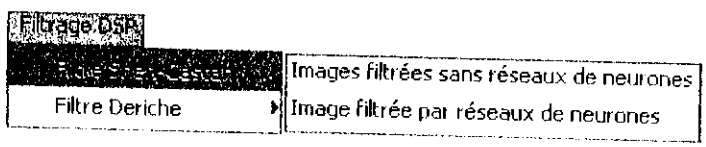
Figure VI.5 : Boîte de dialogue d'introduction de la valeur alpha du filtrage optimal.

VI.2.2.6 Menu Filtrage sous Borland C



Ce menu se compose de deux sous menus déroulants : le filtre de Shen-Castan et le filtre de Deriche. Permettent de visualiser les matrices résultantes des tests faits sous le Borland C sous la forme d'une image.

VI.2.2.7 Menu Filtrage DSP



Ce menu se compose aussi de deux sous menus déroulants : le filtre de Shen-Castan et le filtre de Deriche, qui à leur tour se composent d'autres sous menus permettant de visualiser les matrices filtrées par le DSP sans et avec l'utilisation des réseaux de neurones.

VI.2.2.8 Menu À propos

L'à propos donne des informations sur notre interface

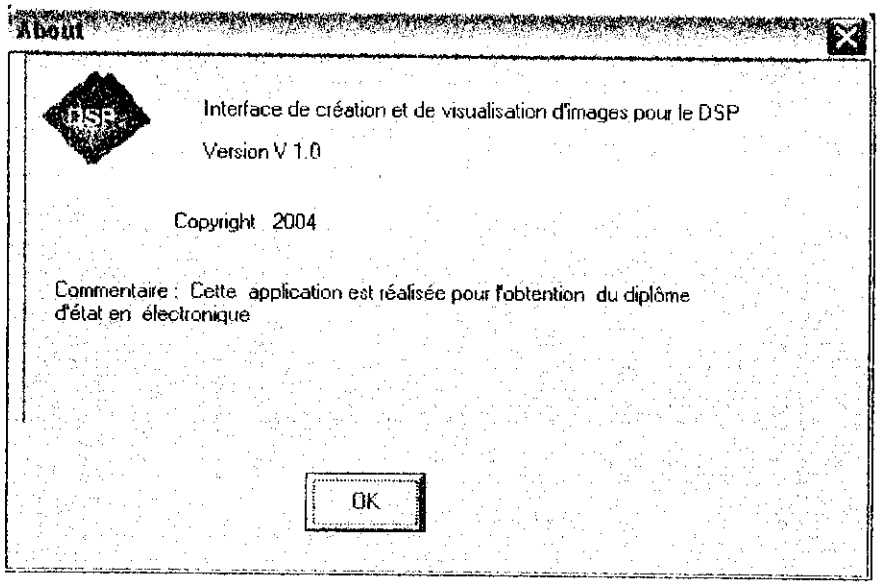


Figure VI.6 : Boîte « à propos ».

VI.2.3. La barre des boutons

Afin d'exécuter les différentes commandes de la barre des menus d'une façon plus rapide, nous avons symbolisé ces dernières à l'aide de dessins dans la barre des boutons : un simple click avec la souris, permet d'exécuter la commande associée.

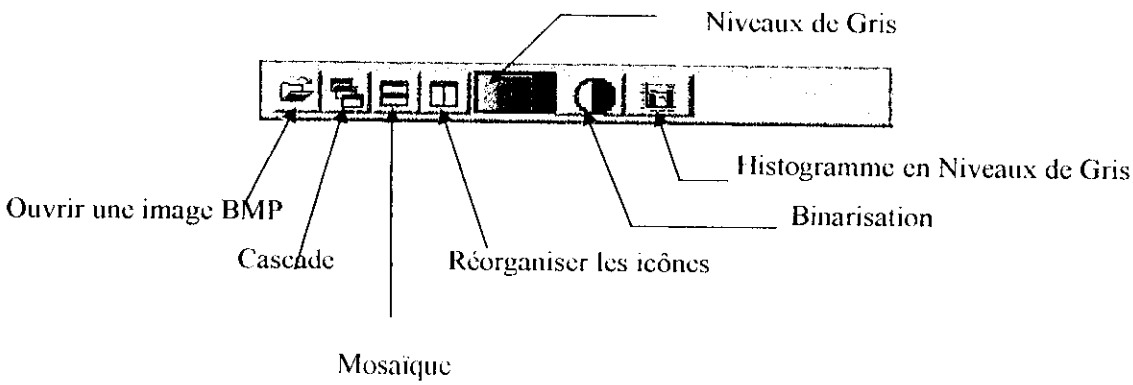


Figure VI.7 : Barre des boutons.

VI.2.4. La barre d'état

C'est une barre qui visualise le nom du traitement en cours : aide des commandes.

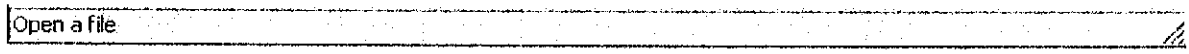


Figure VI.8 : Barre d'état.

VI.3. Conclusion

A travers ce dernier chapitre nous avons donné une description détaillée de l'interface utilisée au cours de notre travail et que nous avons jugée utile pour la création et la visualisation des entrées/sorties du DSP.

Conclusion générale

L'objectif de notre travail était d'étudier l'implémentabilité des filtres optimaux (Shen-Castan et Deriche) sur DSP appliqués à des images. Puis de voir l'influence des réseaux de neurones sur ces derniers. Pour ceci nous avons procédé comme suit :

Tout d'abord, nous avons scanné l'image, puis nous l'avons transformée en niveaux de gris et à l'aide d'une interface réalisée sous Builder C++, nous avons créé les fichiers matrice et dimensions de l'image qui ont servi comme données d'entrée pour le DSP. Cette même interface a servi pour la visualisation des matrices filtrées.
Après avoir testé nos algorithmes sur Borland C, nous les avons transposés sur le simulateur du DSP et nous avons procédé à l'implémentation.
Pour le cas du filtre optimal sans l'utilisation des réseaux de neurones, la configuration de la mémoire du DSP que nous avons établie nous a contraint à traiter l'image par blocs de 128*8 pixels. Ce qui a entraîné des discontinuités et des effets de bord sur l'image filtrée.
En ce qui concerne l'implémentation du filtrage optimal en utilisant les réseaux de neurones, nous avons choisit le réseau multicouche. Son apprentissage a été fait sous Matlab 6.5.
Dans ce cas nous avons pu traiter la totalité de l'image (128*128 pixels) sans la diviser. Les résultats obtenus se sont avérés plus intéressants (les détails apparaissent mieux, les contours sont moins flous et plus fins).
Ainsi nous avons constaté que le filtre de Deriche donne de meilleurs résultats que celui de Shen-Castan pour les deux techniques de filtrage.
Par conséquent, nous concluons que les réseaux de neurones ont optimisé (amélioré) en quelque sorte le filtre optimal.
Néanmoins, les temps d'exécution sur le DSP ont été très longs, ceci étant justifié par le fait que nous avons fait une simulation (sans l'aspect temps-réel).

Pour améliorer notre travail, et comme perspectives envisageables, nous proposons :

- De faire l'apprentissage du réseau multicouche sur d'autres images (prototypes) plus riches en information afin de permettre une meilleure généralisation du réseau.

- D'agrandir la taille de la fenêtre de balayage, et donc plus de points dans le voisinage du pixel courant seront pris en compte, cela permet d'avoir des contours plus précis.
- De réviser le fichier de commande du DSP afin de pouvoir exploiter la totalité de sa mémoire, donc de traiter les images entièrement et de taille plus grande.
- Simuler sur un PC ayant une grande fréquence et assez de RAM afin de diminuer le temps d'exécution du traitement. Encore mieux, l'utilisation de la carte DSP qui permet le traitement en temps-réel. Cependant, faute de moyens, nous aurions aimé travailler sur la génération C6000 qui est plus appropriée au signal image grâce à sa taille mémoire considérable, son haut taux de parallélisme et sa bibliothèque riche.

Nous avons pu atteindre par ce travail les objectifs fixés au départ ; cependant nous espérons par l'utilisation des DSPs pour le traitement d'images ouvrir une nouvelle voie dans le laboratoire d'imagerie au sein de notre école. Aussi nous espérons avoir contribué par ce modeste travail à un domaine qui ne cesse d'évoluer et avoir ouvert de nouvelles portes pour nos chercheurs dans une discipline prometteuse et en pleine évolution.

1. Formats de fichiers bitmap

1.1. Le format TIFF (Tagged Image File Format)

C'est certainement le format graphique le plus connu, créé en 1988 par Microsoft et Aldus.

Sa caractéristique première est d'être un format évolutif : le T initial signifie « balise » (ou tag). Il est possible d'indiquer dans l'en-tête du format TIFF que l'on va ajouter une balise contenant telle information à telle adresse du fichier.

Cette spécificité est à la fois un avantage est une limite : il peut arriver qu'un logiciel ne puisse ouvrir un fichier TIFF contenant un nouveau tag.

Par contre il offre l'avantage d'occuper moins d'espace disque, grâce à son propre algorithme de compression appelé LZW. Actuellement, c'est le choix à privilégier en PAO mais c'est aussi un choix à éviter pour le Web puisqu' aucun navigateur Web ne le lit directement.

1.2. Le format GIF (Graphical Interchange Format)

Le format GIF est un format qui a ouvert la voie à l'image sur le World Wide Web. Conçu à l'origine par la compagnie H&R Block, la renommée du format GIF est due au réseau COMPUSERVE. C'est un format de compression qui n'accepte que les images en couleurs indexées codées sur 8 bits, soit en 256 couleurs. Les images RGB ou CYMK en milliers de couleurs doivent d'abord être converties en 256 couleurs avant d'être exportées en format GIF.

Avantages et caractéristiques : Compression automatique des documents; les fichiers sont alors relativement petits, résolution tonale petite, utilisant un système 8 bits pour enregistrer la couleur (256 couleurs maximum), maximisant encore plus la compression et réduisant la taille des fichiers, facilité de décompression et d'affichage sur des ordinateurs de plates-formes différentes.

1.3. Le format JPEG (Joint Photographic Experts Group)

Les images JPEG sont des images de 24 bits. C'est-à-dire qu'elles peuvent afficher un spectre de 16 millions de couleurs. C'est la meilleure qualité d'images disponible. Par contre, si la carte graphique de l'ordinateur est ajustée à 256 couleurs on ne peut afficher plus de 256 couleurs, les images JPEG auront une moins bonne mine que les

images en format "GIF", qui sont pourtant des images en 8 bits (256 couleurs). Ce format accepte les images RGB et CYMK. Il est sans doute le mode de compression le plus efficace qui soit, avec un bon compromis entre gain d'espace disque, temps de compression/décompression et qualité des images. Ainsi une image brute de 2Mo n'occupera après conversion en JPEG que 130 à 400 Ko selon la qualité d'image voulue. L'inconvénient des images JPEG, c'est qu'elles ne peuvent être importées directement dans un logiciel de mise en page. Le format JPEG est un des formats les plus utilisés dans le monde du World Wide Web (WWW). Il peut être lu directement par les principaux outils de navigation pour le WWW.

Avantages et caractéristiques : Excellent taux de compression, grande résolution tonale, utilisant un système 24 bits pour enregistrer la couleur (16.777.216 couleurs), excellent pour comprimer de gros fichiers images.

1.4. Le format BMP (bitmap)

Le format BMP est le format par défaut du logiciel Windows, il est capable de coder les images jusqu'en 24 bits (16,7 millions de couleurs).

À l'origine, ce format ne comporte aucune compression. Pourtant, en option, une compression RLE (Run-Length Encoding) peut être appliquée ; il s'agit d'une compression sans perte de données, qui encode les répétitions de teintes.

Ainsi, plus une image comporte d'aplats de couleurs uniformes, plus la compression sera efficace et la taille du fichier réduite. Ce type de compression est particulièrement efficace sur les captures d'écran ou les photos comportant, par exemple, un ciel immaculé.

En revanche, elle s'avère limitée pour réduire la taille d'une image comportant du bruit ou une foule de détails. Dans des cas extrêmes, la compression peut même se révéler nulle.

Enfin, l'emploi de ce type de compression sans perte d'information provoque un ralentissement assez sensible lors de l'ouverture et de la sauvegarde d'un fichier. Il s'agit par conséquent, d'options utilisées essentiellement pour archiver une image sans la dégrader ou pour la transférer sur un support amovible comme la disquette.

2. Formats de fichiers vectoriels

2.1. Le format CGM (computer graphic Metafile)

C'est un format de fichier capable de contenir du vectoriel et du bitmap. Créé par l'ANSI, ce format est un standard. Il code les couleurs en RVB : il n'est donc pas destiné au domaine de l'impression offset.

Il n'y a pas d'option spéciale d'enregistrement.

Ce format est assez utilisé en infographie. Son domaine de prédilection est plutôt l'échange de fichiers techniques (type DAO) entre différents logiciels. Il est toutefois lu et écrit par les principaux logiciels de dessin vectoriel et lu, mais non écrit, par certains logiciels de retouche d'images.

2.2. Le format WMF (Windows Metafile)

Le format WMF correspond à des fichiers qui peuvent également contenir des données vectorielles et matricielles.

Créé par Microsoft, il constitue un standard de fait.

Ce type de fichier code les couleurs en RVB : il n'est donc pas destiné à l'impression offset.

Il n'y a pas de compression de données.

2.3. Le format EPS (Encapsulated PostScript)

Il s'agit aussi d'un format *metafile* créé par Adobe.

Le format EPS est extrêmement complet :

- Il accepte les couleurs en RVB et CMJN.
- Il intègre une prévisualisation de l'image.
- Il n'y a pas de système de compression intégré.

Son inconvénient est que la taille des fichiers générés est très importante, et qu'il présente quelques problèmes de compatibilité dus aux niveaux d'évolution du langage.

2.4. Le format DCS (Desktop Color Separation)

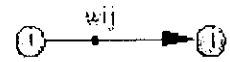
C'est une variante du format EPS très utilisée dans les arts graphiques. Il a été développé par Quark (le créateur de Xpress).

Son principal avantage est qu'il est idéal pour le transfert d'images comportant des fonctions sophistiquées. Son inconvénient réside à la grande taille des fichiers.

1. Règle de Hebb

La première règle d'apprentissage connexionniste fut non supervisée, elle est inspirée de la biologie. En effet vers la fin des années 40, Hebb propose l'idée que le cerveau s'adapte à son environnement en modifiant l'efficacité des connexions entre neurones. Cette modification serait proportionnelle à l'activité simultanée des neurones reliés par la connexion. L'hypothèse est ici qu'une synapse améliore son efficacité seulement quand l'activité de ses deux neurones est corrélée. Le principe de Hebb demeure aujourd'hui biologiquement plausible, et la modification du poids synaptique qu'il décrit a été formulée mathématiquement de la façon suivante :

$$W_{ij} = \eta . a_i . a_j$$



Où :

- h ($0 < h << 1$) est une constante quelconque.
- W_{ij} représente le poids synaptique entre les neurones i et j.
- $\overline{a_i . a_j}$ représente la corrélation entre l'activation des neurones i et j du réseau.

Appliquée aux réseaux neuromimétiques, cette règle simple pousse les neurones à corréler leurs activations avec les valeurs des patrons d'entrée. Un des intérêts d'un tel comportement réside dans le cadre des mémoires associatives, et plusieurs modèles connexionnistes ont été proposés avec des variantes de l'apprentissage hebbien.

La corrélation d'activation peut avoir des effets utiles. Cependant, il n'en demeure pas moins qu'il soit un critère d'apprentissage fort limité, car il ne tient pas compte de la nature de la tâche qu'il doit effectuer.

2. Règle du delta (ou règle de Widrow-Hoff)

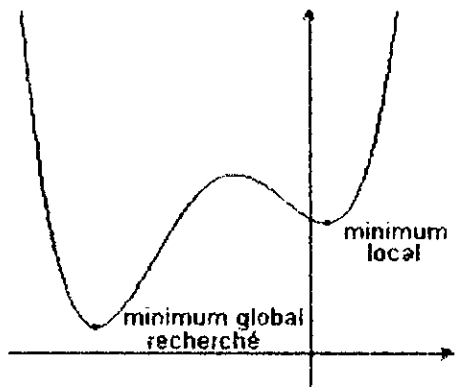
Son but est de faire évoluer le réseau vers le minimum de sa fonction d'erreur (erreur commise sur l'ensemble des exemples). Elle est utilisée dans le modèle d'Adaline (ADAptive LINear Element). L'apprentissage est réalisé par itération (les poids sont modifiés après chaque exemple présenté), et on obtient le poids à l'instant t+1 par la formule : $W(t+1) = W(t) + n.(T - O).E$ si W est le poids, T la sortie théorique et O la sortie réelle, E l'entrée et n un coefficient d'apprentissage (entre 0 et 1) que l'on peut

diminuer au cours de l'apprentissage.

C'est en fait un cas particulier de l'algorithme de rétropropagation du gradient.

3. Rétropropagation du gradient (backpropagation)

Cette règle, utilisée par les réseaux multicouches, consiste simplement en une descente du gradient, qui est une méthode d'optimisation universelle. On cherche à minimiser une fonction d'énergie d'erreur (qui représente l'erreur entre la sortie désirée et la sortie obtenue), en suivant les lignes de plus grande pente. Une fonction d'erreur rapportée à une dimension peut se représenter ainsi :



On peut représenter la descente du gradient comme une bille que l'on poserait sur la courbe, et qui descendrait logiquement la pente (le gradient est la dérivée en plusieurs dimensions et représente donc la pente de la courbe).

La rétropropagation de base utilise le gradient de l'erreur globale (obtenue avec tous les exemples), mais a l'inconvénient de s'arrêter dans le premier minimum local rencontré.

Diverses améliorations ont été apportées :

- **La descente stochastique :** comme dans la règle de Widrow-Hoff, on minimise itérativement l'erreur due à chaque exemple. Les apprentissages de chaque exemple s'influençant les uns les autres, cela permet de passer sur des petits minima locaux. Une présentation aléatoire des exemples donne généralement de meilleurs résultats.
- **La descente avec inertie :** on introduit un moment d'inertie (terme de momentum), qui correspond dans notre image de la bille, à l'inertie qu'elle acquiert en descendant la courbe : son élan lui permettra de ne pas s'arrêter dans le premier minimum local rencontré.
- **Le gradient conjugué**, la **QuickProp**, la **RPROP** sont d'autres améliorations de cet algorithme.

1. Diagramme de développement

Le diagramme de développement nous aide à comprendre l'utilisation des différents composants du Code Composer Studio IDE.

Le CCS étend les outils basiques de génération de code avec un ensemble de possibilité de débogage et d'analyse en temps-réel. Il comprend les différentes phases du cycle de développement suivant :

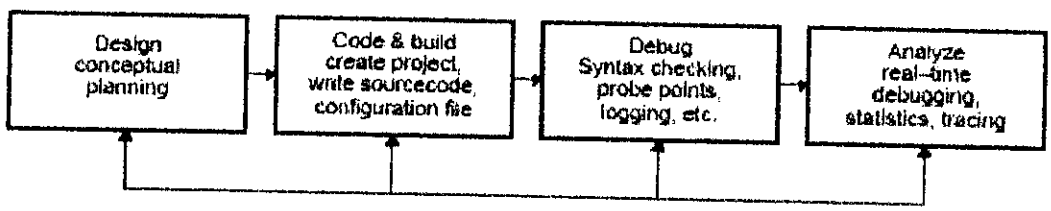


Figure 1 : Diagramme de développement du CCS IDE.

2. Création d'une configuration du système

Le CCS Setup, nous permet de configurer le logiciel CCS IDE afin de travailler sur différents matériels ou cibles du simulateur.

En effet, le CCS Setup dispose de plusieurs configurations des deux familles C55x et C54x et une configuration du système est exigée avant de créer une quelconque application, afin de déterminer quels outils seront utilisés par le CCS.

Dés l'installation du CCS, une configuration du simulateur C55x est fournie par défaut.

Pour créer une nouvelle configuration du système on suit les étapes suivantes :

- **Etape 1 :** On ouvre le CCS Setup en double cliquant sur l'icône du CCS Setup localisée sur le bureau.
- **Etape 2 :** On enlève l'ancienne configuration (ou celle installée par défaut) en cliquant sur le bouton « Clear » (effacer) de la boîte de dialogue « Import Configuration ».
- **Etape 3 :** On clique ensuite sur le bouton « Yes » pour confirmer la commande « Clear ».
- **Etape 4 :** On sélectionne la configuration standard qu'on veut installer sur la liste « Available Configuration », exemple : C5401 Simulator, C5410 XDS510 Emulator, C5510 Simulator, ou C55x XDS510 Emulator...etc.

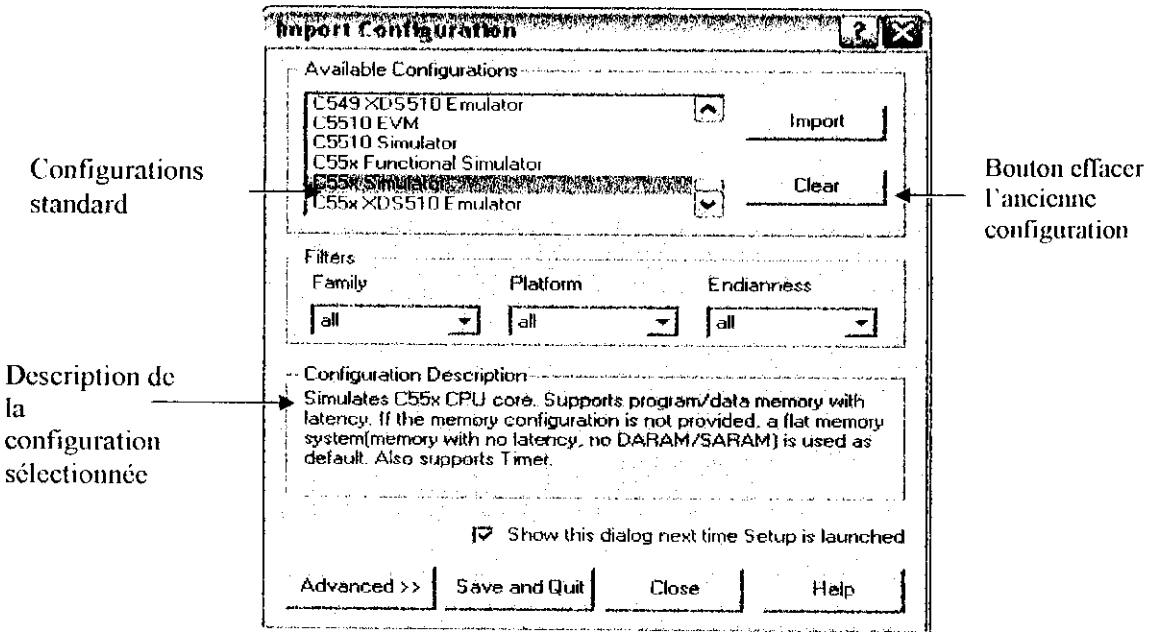


Figure 2 : Boîte de dialogue du bouton « Import Configuration ».

En sélectionnant une configuration standard, une information sera affichée afin de nous aider à déterminer la configuration voulue pour notre système.

Si aucune configuration ne nous convient, on peut créer une configuration personnalisée (voir l'aide en ligne ou les TP fournis par le produit CCS).

- **Etape 5 :** On clique sur le bouton « Import » pour importer notre configuration sélectionnée. Si cette dernière a plus d'une cible on répète les étapes 4 et 5 jusqu'à ce que nous sélectionnons une configuration pour chaque cible.
- **Etape 6 :** On clique sur le bouton «Save and Quit » pour sauvegarder la nouvelle configuration.
- **Etape 7 :** On clique sur le bouton « Yes » quand le message suivant apparaît : « Start Code Composer studio on exit ». Le CCS Setup sera fermé et le CCS IDE s'ouvre automatiquement utilisant la nouvelle configuration créée.

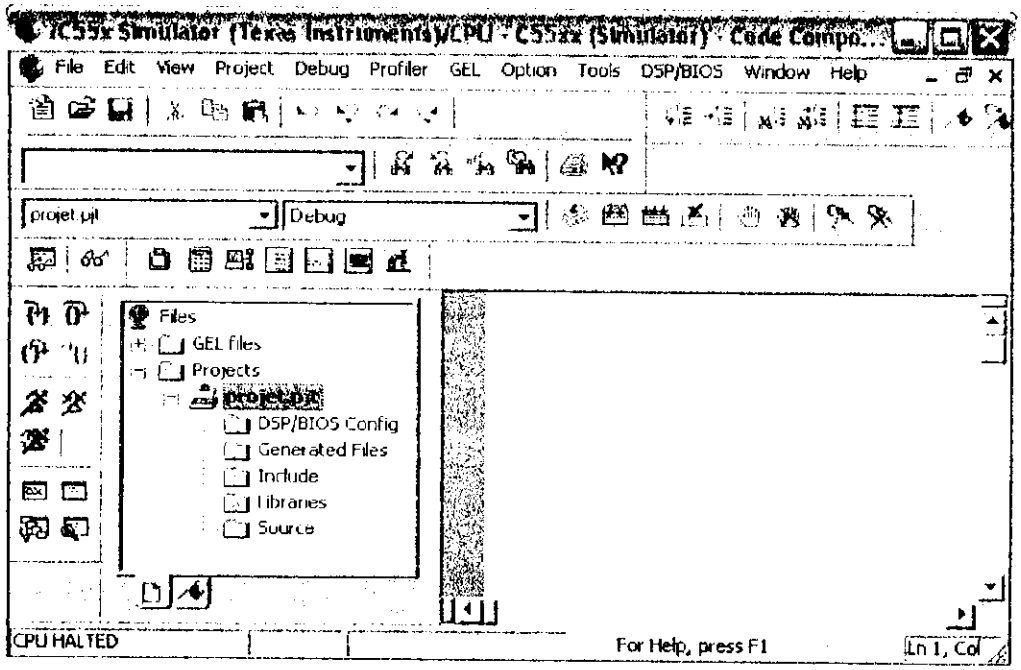


Figure 3 : Interface du CCS IDE.

Maintenant, on peut commencer à créer un nouveau projet dans le CCS IDE.

3. Accès aux TP du CCS (Tutorial)

Afin de se familiariser d'avantage avec l'environnement CCS IDE, il est recommandé d'exécuter les travaux pratiques (Tutorial) proposés par ce dernier. Ceci nous permet de réduire notre temps d'apprentissage, et de prendre connaissances des principales procédures fondamentales.

Pour accéder aux TP du CCS on suit ces deux étapes :

- **Etape 1 :** On ouvre le CCS IDE en double cliquant sur l'icône « CCS2 » localisée sur le bureau.
- **Etape2 :** Sur le menu « Help » du CCS on sélectionne « Tutorial ».

4. Accès à la documentation du CCS

L'aide en ligne fournie par le CCS nous permet d'accéder à la documentation, pour cela il suffit d'ouvrir le CCS Help et de sélectionner Help → Contents, une page de bienvenue (Welcome) apparaît contenant les différents liens utiles pour accéder à la documentation.

Remarque : On peut accéder à la documentation via le menu de démarrage en cliquant sur Démarrer → Programme Files → Texas Instruments → Code Composer Studio 2 → Documentation.

5. Gestion d'un projet

5.1. Création d'un nouveau projet

Les procédures suivantes nous permettent de créer de nouveaux projets, un ou plusieurs à la fois, seulement le nom du dossier de chaque projet doit être unique. L'information pour un projet est enregistrée dans un dossier sous la forme (*.pjt).

Note : On peut ouvrir plusieurs projets en même temps.

Les étapes de la création d'un projet se déroulent comme suit :

- **Etape 1 :** Sur le menu du CCS IDE on sélectionne Projet → New, la fenêtre ci-dessous apparaît :

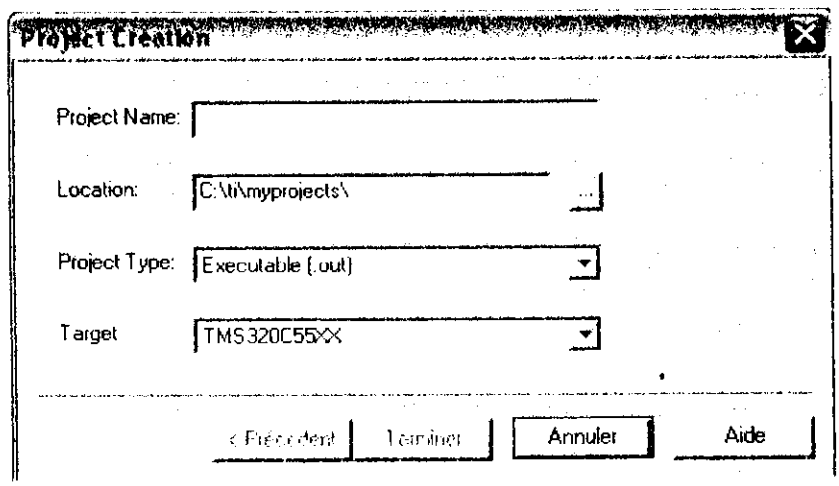


Figure 4 : Boîte de dialogue de « Project Creation ».

- **Etape 2 :** Sur le champ « Project Name » on écrit le nom qu'on veut attribuer à notre projet.
- **Etape 3 :** Sur le champ « Location » on choisit le chemin ou on veut enregistrer notre projet.
- **Etape 4 :** Sur le champ « Project Type » on choisit le type du projet dans la liste proposée, telle que : Exécutable (.out) pour que le projet produit un dossier exécutable ou Bibliothèque (.lib) pour construire une bibliothèque de l'objet.
- **Etape 5 :** sur le champ « Target » on choisit la famille qui identifie notre CPU. Cette information s'avère très nécessaire lorsque les outils sont installés pour plusieurs cibles.
- **Etape 6 :** on clique sur « Finish », le CCS IDE crée le dossier de notre projet (projectname.pjt) qui contiendra tous les dossiers utiles pour ce dernier. Le nouveau

projet devient automatiquement le projet actif et aura les options de débogage et de liens (linker) fournies par TI.

Après avoir créé notre projet on ajoute à sa liste : le fichier code source, bibliothèque et le fichier de commande.

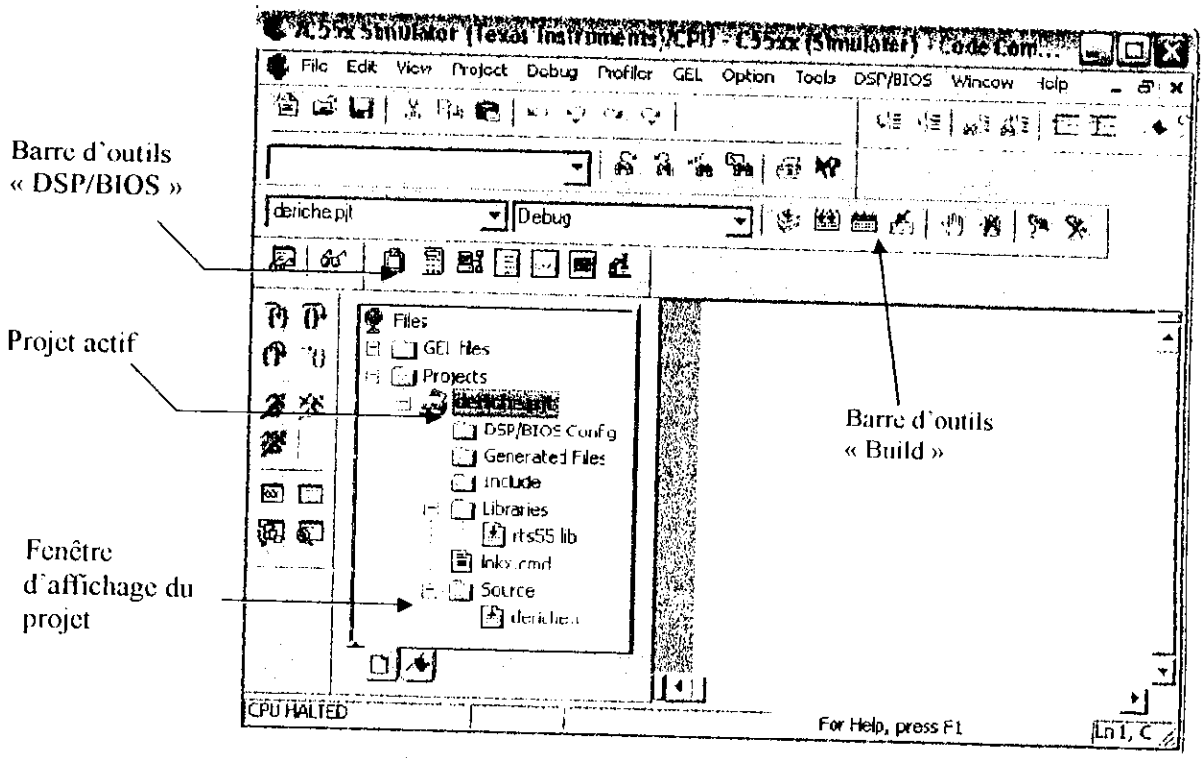


Figure 5 : Fenêtre de base du CCS.

5.2. Ajout de fichiers à un projet

On peut ajouter plusieurs et différents fichiers à notre projet. Les types sont montrés dans la boîte de dialogue (figure 6). La procédure à suivre pour ajouter des fichiers à notre projet se déroule comme suit :

- **Etape 1 :** on sélectionne Project → Add Files to Project, ou on clique par le bouton droit sur le nom du projet affiché sur la fenêtre puis on sélectionne « Add Files». La boîte de dialogue suivante apparaît :

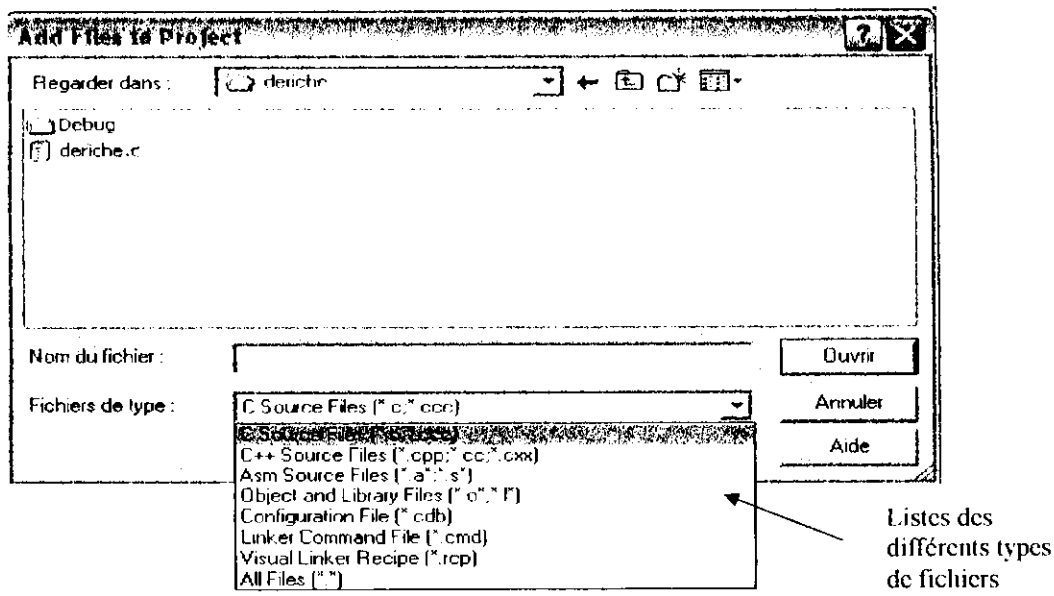


Figure 6 : Boîte de dialogue de « Add Files to Project».

- **Etape 2 :** dans cette boîte de dialogue on spécifie le fichier qu'on désire ajouter. Si le fichier n'existe pas dans le répertoire courant on parcourt jusqu'à trouver son chemin.

Note : Ne jamais essayer d'ajouter manuellement les fichiers en-têtes/Include (*.h) au projet, ces fichiers seront ajoutés automatiquement lorsqu'on clique sur «Scan All Dependencies »

- **Etape 3 :** on clique sur « Open » pour ajouter le fichier spécifié au projet.

L'affichage du projet est automatiquement mis à jour lorsqu'un fichier est ajouté.

Le manager du projet organise les fichiers dans les dossiers : Source, Include, Librairies, et DSP/BIOS Configuration.

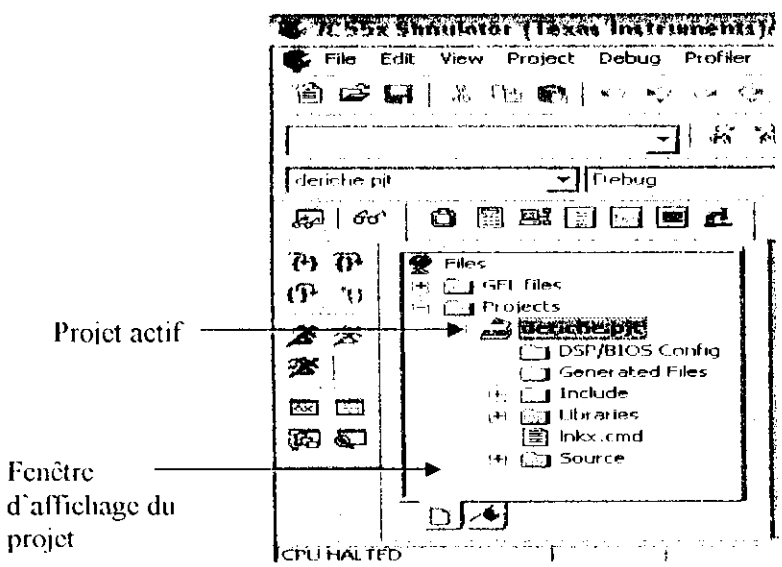


Figure 7 : Affichage du projet.

Note : Si on veut enlever un fichier du projet on clique sur ce fichier par le bouton droit et on sélectionne sur le pop-up menu « Romove from project ».

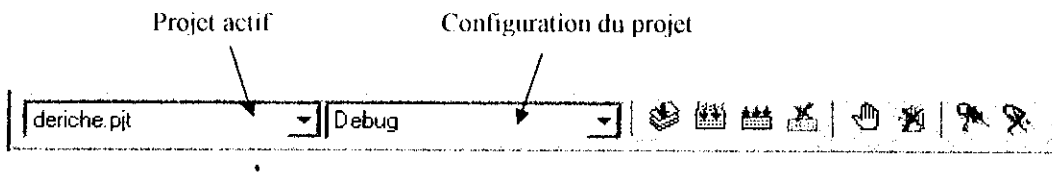
5.3. Exécution du programme (Building and Running)

Pour exécuter un programme on suit les étapes suivantes:

- **Etape 1 :** On sélectionne Project → Rebuild All ou on clique dans la barre d'outils sur le bouton .

Le CCS IDE recompile, réassemble, et relie tous les fichiers dans notre projet. Des messages sur la progression du processus d'exécution sont affichés dans un cadre au fond de la fenêtre.

- **Etape 2 :** Par défaut, le fichier exécutable « .out » est construit dans le dossier « Debug » du projet courant, pour changer cette location on sélectionne une location différente dans la barre d'outils du CCS.



- **Etape 3 :** On sélectionne File→ Load program, ensuite on sélectionne le programme que nous venons de créer (.out), et on clique sur « Open ». Le CCS IDE charge le programme sur la cible DSP et ouvre une fenêtre « Dis-Assembly » qui montre les instructions traduites en assembleur composant le programme.
- **Etape 4:** On choisit View/Mixed Source/ASM, cela permet de voir notre code source écrit en C et sa traduction en assembleur simultanément.
- **Etape 5 :** On sélectionne Debug→ Go to Main pour commencer l'exécution à partir de Main (ie programme principal). L'exécution s'arrête au Main et est identifiée par la flèche .
- **Etape 6** : On sélectionne Debug→ Run ou on clique sur le bouton (bouton sur la barre d'outils).
- **Etape 7 :** On sélectionne Debug→ Halt pour arrêter l'exécution du programme

6. Chargement et sauvegarde des données

Le CCS nous permet de charger et de stocker des données sous format « *.dat » par trois différents moyens :

● En utilisant Load/Save à partir du menu :

On sélectionne File→ Data→ Load pour charger les données manuellement à partir d'un fichier « *.dat » qui doit contenir une ligne d'en-tête du DSP formée de 5 champs :

- Le *nombre « magique »* fixé à 1651 ;
- Le *format* qui est un entier entre 1 et 4, spécifiant le format des données dans le fichier (1= hexadécimal, 2= entier, 3= entier long, 4= nombre flottant);
- L'*adresse de départ* du bloc qui a été sauvegardé ;
- Le *numéro de page* correspondant à la page source du bloc ;
- La *longueur* du bloc.

Exemple d'une en-tête : 1651 1 0 1 100.

On doit préciser la variable de destination (ou son adresse mémoire) de notre chargement et le nombre d'échantillons à charger.

Pour la sauvegarde des données traitées, on sélectionne File→ Data→ Save, on précise la variable qu'on veut sauvegarder (ou son adresse mémoire) et le nombre d'échantillons à sauvegarder.

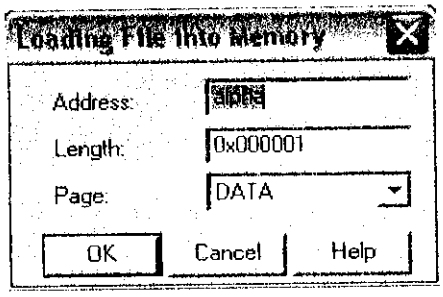


Figure 8 : Chargement de données

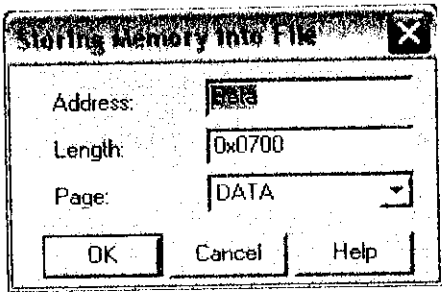


Figure 9 : Sauvegarde de données.

● En utilisant les probe-points (les sondes) :

Ces sondes servent aussi à charger et sauvegarder manuellement les fichiers de donneés input/output (*.dat), elles seront placées sur le curseur positionné sur le code source de notre projet, il suffit de cliquer sur le bouton . Ensuite on clique sur File → File I/O on obtient la fenêtre (figure 10) sur laquelle on choisit les fichiers à

connecter aux points sondes pour le chargement (Input) ou pour la sauvegarde (output).
Ces fichiers doivent contenir l'en-tête expliquée précédemment.

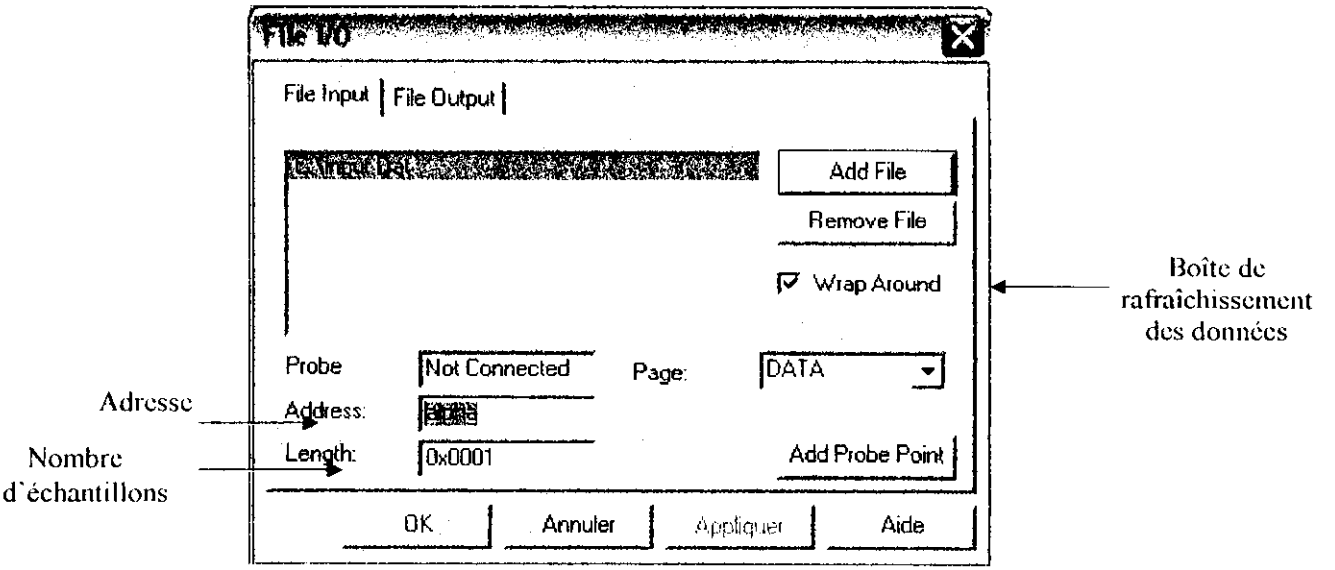
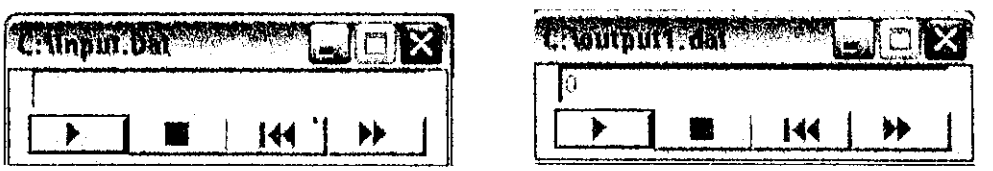
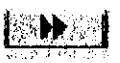


Figure 10 : Boîte de dialogue « File I/O ».

Pour chaque sonde connectée à un ficher on obtient les icônes suivantes :



Il suffit de cliquer sur  pour charger ou sauvegarder les données.

Remarque : si notre fichier source contient des données qui se répètent dans le temps, on sélectionne l'option « wrap around » sur la boîte de dialogue, par exemple cas d'un signal sinusoïdal.

- **En utilisant les instructions de la directive « stdio.h » dans le code source:**

Les instructions : fopen, fscanf, fprintf, fclose,..etc, servent à charger et sauvegarder automatiquement les fichiers de données input/output (*.dat). Ces fichiers lus ne doivent pas contenir l'en-tête du DSP.

Glossaire

- **Neurone :** Unité de traitement de l'information dans le cerveau, au nombre mille milliards environ.

- **Base d'apprentissage :** Exemples utilisés pour l'apprentissage, représentatifs du comportement de la fonction à réaliser.

- **Base de test :** Exemples différents de ceux de la base d'apprentissage, utilisés pour mesurer les performances de la généralisation.

- **Généralisation :** Principal intérêt des réseaux de neurones artificiels. Capacité à répondre correctement à des situations inconnues (càd n'apparaissent pas dans la base d'apprentissage).

- **Mémoire associative :** Mémoire adressable par son contenu, à la différence des mémoires plus classiques où l'adresse est requise.

- **Connexionnisme :** Discipline définie par l'utilisateur des réseaux de neurones artificiels.

- **Supervisé :** Les poids sont modifiés en fonction de la sortie désirée.

- **Non supervisé :** Les sorties désirées sont inconnues, les entrées sont projetées sur l'espace du réseau.

- **Code Composer Studio :** Un système intuitif d'outils de développement intégrés dans un environnement facile à utiliser. L'analyse en temps-réel et la possibilité de visualisation des données, l'architecture ouverte et des outils de génération du code évolués réduisent la complexité du développement sur DSP. Ils nous permettent de centraliser nos ressources et créativités en ajoutant de la valeur à notre application.

- **BIOS :** Une partie d'un noyau qui offre l'interface la plus basse aux périphériques. Dans le but d'offrir une performance acceptable, les vendeurs de software peuvent accéder aux routines dans le BIOS.

- **Emulation :** Le processus de brancher une cible dans le but du débogage et l'analyse en utilisant une combinaison d'outils software et hardware.

- **Un hôte :** Un ordinateur connecté à un réseau.

- **Un PC hôte :** Une station de développement.

- **Un IDE (Integrated Developpement Environnement) :** Un système pour supporter le processus d'écriture de software.

- **JTAG :** Une méthode de test d'un circuit avec un accès limité aux signaux internes avec scannage de contours.

- **Librairie :** Une collection de fonctions software ou modules qui peuvent être automatiquement incluses à l'intérieur de l'application quand elle est créée (création de l'exécutable). Les programmeurs souvent placent des parties de code réutilisable dans des librairies ou elles sont facilement accessibles.

- **Linker (éditeur de liens) :** Un programme du PC qui accepte des fichiers code objet d'un ou plusieurs différents modules du programme compilé, et les lier (édition de liens) ensemble en un programme exécutable, qui résout les références d'un module à un autre.

- **Page :** Le DSP divise sa mémoire en plusieurs blocs de 128 mots qu'on appelle page, afin de diminuer la taille du code en mémoire.

Bibliographie

[1] André MARION, « Introduction aux techniques de traitement d'images », Edition : EYROLLES, Paris, 1987.

[2] Gérard MICHEL-DUTHEL, « Images numériques & Formats graphiques», Edition : Compus press, Paris, 2000

[3] Jean FRUITET, « Outils et méthodes pour le traitement des images par ordinateur »,
URL: http://etudiant.univ-mev.fr/~fruitet/Trait_image/an_image.html

[4] C. BOUDRY, « Qu'est ce qu'une image numérique »,
URL: http://www.ecr.jussien.fr/urfist/image-numérique.htm
URFIST de Paris, 2100-2002

[5] Gaël JABOULAY, « Format de fichiers d'images »,
URL : http://tecfa.unige.ch/tecfa/teachg/staf13/fiches-mm/format fichier.htm

[6] KADDOUR Chakib, « Géneralités sur le traitement d'images », 1999
URL: http://iquebec.ifrance.com/kadchakib/chap1/chap1.htm

[7] J-P. COCQUEREZ, « Analyse d'image : Filtrage et segmentation », Edition : MASSON, 1995.

[8] S. DOUMANDJI- Y. BOULFANI, « Système de vision par ordinateur », Séminaire, ENP 2002-2003

[9] R. HAROUN- S. A. BOUHRICHE, « Séparation des chromosomes se touchant ou se chevauchant a partir de Meta phases », diplôme d'ingénieur d'état en Electronique, ENP- Promotion 2001-2002

[10] H.BENHIDOUR, « Système de classification des chromosomes par réseau de neurones », diplôme d'ingénieur d'état en informatique, option : Systèmes Informatiques, INI, Promotion 2000-2001.

[11] Y. BRIKI – D. SMATI, « Contribution à la conception d'un outil d'aide pour le choix d'une technique de traitement d'images », diplôme d'ingénieur d'état en informatique, option : Systèmes Informatiques, INI- Promotion 2002-2003.

[12] C. BOUDRY, « Amélioration des images par filtrage », URFIST de Paris, 2100-2002
URL: http://www.ecr.jussien.fr/urfist/image-numérique.htm

[13] Julien FAUQUEUR, « Traitement et manipulation d'images », 2000-2001
URL: http://www.rocq.inria.fr/~fauqueur/java/sujet/projet.html

[14] Pierre-Arnaud CHASTANET, «Filtre de Shen ou de Deriche pour détecter un contour ?», 1998
URL: http://www.tsi.enst.fr/tsi/enseignement/ressources/mti/Shen-ou-Deriche/node25.html

[15] Brun LUC, « Les opérateurs optimaux », 2004
URL: http://www.univ-reims.fr/Labos/LERI/membre/luc/ENSEIGNEMENT/COURS/TR_IMG/node83.html

[16] Philippe DAX, « Langage C », Edition: EYROLLES, Paris, 1984.

[17] Y. BOULFANI, « Réseaux de neurones et leurs applications », Mini projet, ENP-2003-2004

[18] A. RAHMANIA, « Système de détection d'aberrations chromosomiques structurales », diplôme d'ingénieur d'état en informatique, USTHB- Promotion 2002-2003

[19] S. BENDI- D. DOUACHE, « Système de reconnaissance de texte manuscrit arabe », diplôme d'ingénieur d'état en Electronique, ENP- Promotion 2002-2003

[20] Jeanny HERAULT- Christian JUTTEN, « Réseaux neuronaux et traitement du signal », Edition : HERMES, Paris 1994

[21] Y. YACTINE, « Réseaux Neuronaux », 1999

URL: http://www.cict.fr/cict/personnel/stpierre/reseaux-neuronaux/node1.html

[22] F.DEVAUX, « Filtrage d'image par réseaux de neurone », Département MIME, Université Paris 8, 1997

[23] Techniques de l'Ingénieur, Traité d'Electronique, E3565 « Processeurs de traitement numérique du signal (DSP) ».

[24] Geneviéve BAUDOIN- Ferial VIROLLEAU, « DSP Les processeurs de traitement du signal, famille TMS320C54x», Edition: DUNOD, Paris 2000.

[25] Geneviéve BAUDOIN- Ferial VIROLLEAU, « DSP Les processeurs de traitement du signal, famille TMS320C54», Edition: DUNOD, Paris 1996.

Documents PDF de Texas Instruments : www.ti.com

[26] SPRU393.pdf, «TMS320C55x Technical Overview».

[27] SPRU307a.pdf, « TMS320C54x DSP Functional Overview ».

[28] SPRU312.pdf, «TMS320C55x DSP Functional Overview».

[29] SPRU368d.pdf, «DSP Starter Kit».

[30] SPRU509c.pdf, «Code Composer Studio Getting Started Guide».

[31] SPRU035.pdf, «TMS320C3x/C4x Assembly Language Tools User's Guide».

[32] SPRU280d, «TMS320C55x Assembly Language Tools User's Guide».

[33] S. DOUMANDJI, « Les DSPs Processeurs de traitement du signal», Mini projet, ENP 2003-2004

[34] N. MAZOUZ, « Processeurs de traitement digital du signal/DSP/TMS320Cxx », Séminaire, ENP 2003-2004

[35] M.S.TOUHAMI & A.BOUGHRIRA, « Processeurs de traitement de signal (DSP) TMS320C31 », Séminaire, ENP 2002-2003

[36] Noureddine BENABADJI- Nour el islam BACHARI, « Guide pratique de programmation en langage C », Edition : HOUMA, Alger, 2000.

[37] Philipe SPOLJAR, « Borland C++ Builder », Edition: SYBEX, 1997.

[38] Ken HENDERSON- Kent REISDORPH, « Le programmeur C++ Builder » Edition: BORLAND Press, SAMS Publishing, Paris, 1997.

Documents extraits du site : www.xcotton.com

[39] Tome 1, « Généralités sur les DSP ».

[40] Etienne SICARD- Sonia DELMAS-BENDIA, « Une introduction aux DSP », Proposition au bulletin de l'Union des Physiciens.
URL: http://intrage.insa-tlse.fr/~dsp/DSPf.pdf

[41] CNAM, « TEXAS-TMS320C50 ».

[42] Cours_5, « Le traitement et l'analyse des images numériques ».